Universidad de las Ciencias Informáticas

Facultad 5

Módulo de Interoperabilidad del Cuadro de Mando Integral para el SCADA Sainux.

Trabajo de diploma para optar por el título de ingeniero en ciencias informáticas.

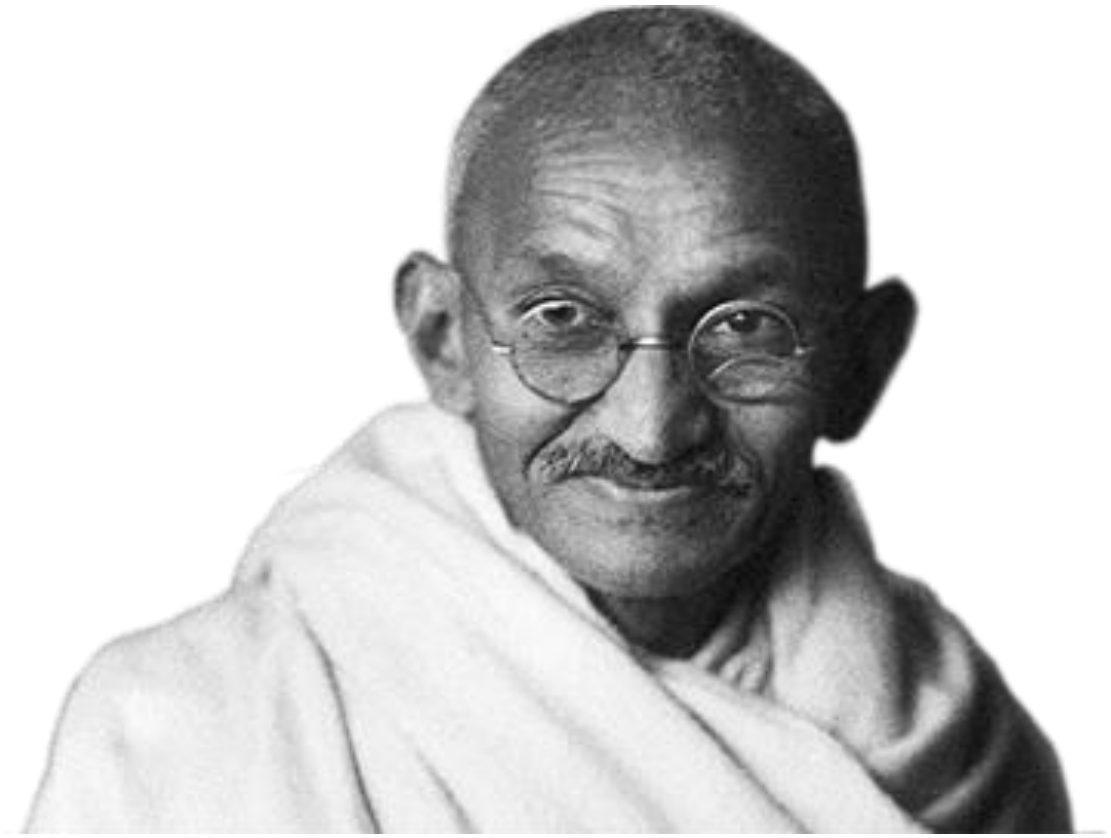
Autor: Felix Alejandro Paez Garriga

Tutor: Ing. Dayrien Corrales Díaz.

Ciudad de la Habana, Cuba

Curso 2015-2016

Pensamiento



"El hombre no es más que el producto de sus pensamientos. Se convierte en lo que piensa". Gandhi.

Dedicado a:

Mi abuela Vilma que, aunque no está presente sé que observa sus frutos desde un lugar mejor, gracias por cuidar de mi desde chiquitico!!!.GRACIAS ABUE!!!.

A mi mamá y papá que me dieron la gracia de vivir y apoyarme en todos mis proyectos de vida y sobre todo dándome todo su amor. GRACIASSSSSSSSSSSS!!!.

A mi familión que en las buenas y malas siempre estuvo apoyándome y aconsejándome sin dudar nunca de mis actitudes. LOS QUIERO CON TODO MI CORAZÓN!!!.

A mi hermanón, que siempre es mi otra mitad y una persona muy importante en mi vida, GRACIAS POR TODO!!!.

Agradecimientos

A mis padres, quienes son mi razón de existir, además de ser fuente de inspiración y superación. Gracias por estar en todo momento aconsejándome y guiándome en mi historia.

A mi hermano, que siempre me apoyó en los tiempos difíciles, cómplice en nuestras historias y es una de las personas que más quiero y admiro.

A mis tías y tíos que siempre me ayudaron y aconsejaron por el buen camino, por mostrar su cariño y dedicación a su sobrino, por no dudar de mis capacidades, gracias!!!.

A mis primas, primos y familia en general por su apoyo y dedicación, gracias!!!.

A mi primo Rafael por proveerme de la tecnología necesaria para el desarrollo de esta obra.

A Yanet y Javier por prestarme su ayuda incondicional.

A mi tutor por darme la oportunidad de desarrollar la presente tesis y guiarme en el desarrollo de la misma.

Al tribunal por sus correcciones, que fueron de mucha ayuda.

A Adolfo por prestarme su ayuda incondicional, gracias por ser compañero y amigo.

A mis compañeros de aula, por compartir la vida universitaria, en especial a Yaser con el que estudié para todas las pruebas, proyectos y salimos satisfactoriamente, gracias amigo.

A los profesores de la UCI que de una forma u otra contribuyeron a mi desarrollo profesional, gracias!!!.

Declaración de autoría.

Declaro ser el autor de la presente tesis: **"Módulo de Interoperabilidad del Cuadro de Mando Integral para el SCADA Sainux"** y reconozco a la Universidad de las Ciencias Informáticas los derechos patrimoniales de la misma, con carácter exclusivo.

Para que así conste firmo el presente a los ____ días del mes de _____ del año 2015.

Firma del autor

Felix Alejandro Paez Garriga

Firma del tutor

Ing. Dayrien Corrales Díaz.

Resumen

La Universidad de Ciencias Informáticas está capacitada para desarrollar productos y servicios informáticos de automatización industrial tanto a nivel nacional como internacional, según el Modelo de Madurez de Capacidad Integrada (CMMI). En dicha universidad radica el Centro de Informática Industrial (CEDIN), el cual es encargado del software SCADA Sainux que no cuenta con las herramientas adecuadas para garantizar la toma de decisiones, lo cual sugiere las prestaciones de un Cuadro de Mando Integral(CMI) basado en módulos distribuidos para generar inteligencia de negocio.

El Módulo de Interoperabilidad para el Cuadro de Mando Integral en el SCADA Sainux cuenta con un servidor Node.js basado en JavaScript para manejar la información por eventos mediante tecnología WebSocket y donde se utiliza el servidor de base de datos MongoDB para lograr una persistencia de los datos. En función de lograr este objetivo se realizó un estudio de diversas tecnologías y herramientas que junto a la metodología AUP-UCI fomentaron las bases para un correcto desarrollo de dicha aplicación.

Palabras claves:

Cuadro de Mando Integral, Interoperabilidad, MongoDB, Node.js, WebSocket.

ÍNDICE

Índice de figuras

Índice de Tablas

Introducción

En la actualidad, para las empresas definir como maniobrar y hacer la información fructífera en función de su calidad, es primordial para que la gestión empresarial se traduzca en resultados convenientes para su desarrollo. Para ello dichas empresas emplean una gama de estrategias y herramientas dirigidas a la administración y creación de conocimiento.

Una de las principales consecuencias de los avances tecnológicos es la cantidad de información de interés generada diariamente, siendo cada vez más importante almacenarla y recuperarla, generalmente esta se encuentra dispersa, poco estructurada e invisible lo que hace muy engorroso el proceso de obtención de la misma e imposibilita el rápido acceso a la información.

El procesamiento de los datos es la técnica que se emplea en la recolección de los datos para que, luego de ser evaluados y ordenados, permitan obtener información útil para que cada usuario interesado en tomar decisiones o trazar estrategias que favorezcan el bienestar de un negocio o de una tarea rutinaria.

Los sistemas informáticos de supervisión, control y adquisición de datos (SCADA, por sus siglas en inglés) se basan en la adquisición de datos de procesos remotos y está diseñado fundamentalmente para funcionar sobre ordenadores en la supervisión y control de la producción, proporcionando comunicación con los dispositivos de campo (controladores autónomos, autómatas programables, etc.) (Monteros, y otros, 2004).

El ambiente de producción donde se despliegan los sistemas SCADA genera grandes volúmenes de datos, no obstante, las interfaces hombre-máquina de los sistemas de supervisión y control se limitan a la representación de una pequeña parte del total de los datos almacenados, para responder así a necesidades puntuales de las operaciones supervisadas desde las consolas. Por ello, existe un alto grado de inutilidad de los datos almacenados y pudieran ser procesados para generar conocimientos relacionados con el proceso de producción y así apoyar la toma de decisiones estratégicas en la industria.

En Cuba la industria del *software* ha aumentado su producción considerablemente, con el objetivo de satisfacer las necesidades de informatización de la sociedad cubana, alcanzando un nivel de competitividad acorde a los estándares internacionales y logrando potenciar las exportaciones de *software*.

La Universidad de las Ciencias Informáticas (UCI) creada con el objetivo formar profesionales altamente calificados en la rama de la Informática; producir aplicaciones y servicios informáticos, a partir de la vinculación estudio-trabajo como modelo de formación y servir de soporte a la industria cubana de la informática (Castro, 2002), cuenta con varios centros productivos entre los cuales se encuentra el Centro de Informática Industrial (CEDIN). Este centro tiene como misión desarrollar productos y servicios informáticos de automatización industrial, con un alto valor agregado y que cumplan las necesidades y expectativas de los clientes, potenciando la formación especializada y la investigación.

El CEDIN desarrolla soluciones que favorecen al sector industrial dentro y fuera del país. Estos tipos de sistemas se encargan de generar datos sobre el estado de los componentes que intervienen en los procesos de producción, los cuales se conocen como: sistemas de procesamiento transaccional en línea (OLTP, por sus siglas en inglés). Entre las soluciones OLTP del CEDIN se encuentra el SCADA SAINUX y las funcionalidades de este no están orientadas a la realización de inteligencia de negocio.

Los productos del CEDIN no están orientados a realizar funciones propias de sistemas de inteligencia de negocio, pues la filosofía de estos últimos se enmarca en el procesamiento analítico en línea (OLAP, por sus siglas en inglés); esto se traduce en que la finalidad de los productos del centro se limita a la generación, recolección y visualización de datos. Por lo cual surge la idea de desarrollar un Cuadro de Mando Integral donde se apliquen nuevas técnicas de análisis de datos históricos basadas en minería de datos que apoyen la toma de decisiones y las proyecciones futuras de las empresas.

Con la intención de desarrollar las funcionalidades principales del cuadro de mando integral, encargadas de recuperar información de bases de datos, aplicar algoritmos de inteligencia artificial a los datos para obtener patrones y relaciones entre ellos y visualizar la información generada; todo ello en un ambiente distribuido se identifican las siguientes características:

1. Se identifican al menos 3 módulos para realizar estas operaciones.
2. Cada módulo puede ser desarrollado con tecnologías diferentes, dependiendo de las bondades que se deseen aprovechar en cada caso.
3. Aun siendo un sistema distribuido, el funcionamiento del mismo debe ser transparente para los usuarios finales.

La característica fundamental de los sistemas de inteligencia de negocio de transformar datos en información, y esta última en conocimiento, no siempre se realiza siguiendo pasos estrictos. Esto quiere decir que entre los módulos el intercambio de mensajes no siempre va a ser unidireccional o con una marcada dependencia directa.

Teniendo en cuenta esta problemática se plantea el siguiente **problema a resolver**:

¿Cómo proporcionar interoperabilidad entre los subsistemas del Cuadro de Mando Integral del SCADA SAINUX bajo una arquitectura distribuida?

Por consiguiente, el **objeto de estudio** es el proceso de interoperabilidad entre subsistemas distribuidos.

En consecuencia, el **campo de acción** es:

Interoperabilidad entre subsistemas distribuidos para apoyar la toma de decisiones.

En vísperas de darle solución al problema planteado se identifica el siguiente **objetivo general**:

Desarrollar un módulo que maneje la interoperabilidad entre los subsistemas del Cuadro de Mando Integral del SCADA Sainux.

Para alcanzar la meta propuesta se proponen las siguientes **tareas de investigación**:

1. Sistematización de los fundamentos teórico-metodológicos para el desarrollo de un módulo del Cuadro de Mando Integral para el SCADA Sainux.
2. Diagnóstico del estado del arte de las principales tecnologías y herramientas para desarrollar aplicaciones que provean interoperabilidad.
3. Análisis de patrones de diseño para la construcción del subsistema.
4. Diseñar la comunicación entre los diferentes subsistemas del CMI.
5. Caracterizar las principales funcionalidades a implementar en el módulo.
6. Implementar las funcionalidades del módulo caracterizado.
7. Validación de las funcionalidades implementadas mediante la realización de pruebas.

Para dar cumplimiento al objetivo se emplearon varios **métodos científicos** de investigación, los cuales se muestran a continuación:

Métodos del nivel teórico:

Analítico-Sintético: Se utiliza para comprender el funcionamiento de la interoperabilidad entre subsistemas distribuidos en función de las características que presenta para trazar las directrices del desarrollo del módulo mediante los conocimientos sintetizados.

Hipotético-Deductivo: Se emplea en la investigación de la interoperabilidad entre subsistemas distribuidos y como a partir de las experiencias obtenidas se deduce una solución informática.

Métodos del nivel empírico:

Observación: Este método será utilizado para poder percibir los cambios presentes durante el proceso de desarrollo en función de combatir las principales deficiencias de la aplicación.

Estructura de la tesis

La tesis consta de introducción, tres capítulos (I- Fundamentación teórica, II-Análisis y diseño del sistema, III- Desarrollo de un módulo de Interoperabilidad y su validación), conclusiones, recomendaciones y bibliografía.

Capítulo I: Fundamentación teórica. En este capítulo se expone los fundamentos teóricos relacionados con el proceso de interoperabilidad entre subsistemas distribuidos, específicamente los relacionados con el apoyo de toma de decisiones.

Capitulo II: Análisis y diseño del sistema. En este capítulo en función del modelo conceptual diseñado se hace un levantamiento de requisitos, donde los requisitos funcionales apoyan su descripción en las historias de usuario. Además, se expone la arquitectura del sistema y los patrones de diseño utilizados, todo enmarcado en la metodología de desarrollo de *software* AUP-UCI.

Capítulo III: Implementación y pruebas del sistema. En este capítulo se ilustra el modelo de despliegue del sistema y las pruebas relacionadas al *software,* específicamente las de caja blanca.

Capítulo I: Fundamentación teórica

1.1 Introducción

En este capítulo se expondrán los fundamentos teóricos relacionados con el proceso de interoperabilidad entre subsistemas distribuidos, específicamente los relacionados con el apoyo de toma de decisiones.

1.2 Interoperabilidad

Las redes de computadoras son típicamente heterogéneas ya que las distintas empresas adquieren múltiples dispositivos informáticos de prestaciones y tecnológicas diferentes. Evidencian este comportamiento los distintos cambios tecnológicos en periodos relativamente cortos. En este marco, estos cambios van aparejados de mejor rendimiento, calidad, seguridad, etc. Además, la diversidad en una red de computadoras puede hacerla más resistente a errores, ya que un problema en una determinada máquina, aplicación o sistema operativo es poco probable que afecte a otros sistemas corriendo en diferentes sistemas operativos y aplicaciones. De esta forma se puede definir la interoperabilidad como la “capacidad que tiene un producto o un sistema, cuyas interfaces son totalmente conocidas, para funcionar con otros productos o sistemas existentes o futuros y eso sin restricción de acceso o de implementación” (1).

1.2.1 Modelo de madurez de la interoperabilidad

El modelo de madurez de interoperabilidad clasifica en cinco niveles, identificando y determinando características en términos de cuatro atributos: Procedimientos, Aplicaciones, Infraestructura y Datos(PAID) como se muestra en la figura 1. (Proal C. 2015)

Naturaleza de la Interacción Operacional de Información	Ambiente de Computación Correspondiente	Código de Nivel	Implicaciones P	A	I	D
Manipulación interactiva de Dominios diferentes	Universal	4	Nivel Empresa	Interactivo	Topologías Múltiples	Modelo Empresa
Aplicaciones y Bases de Datos Compartidas	Integrado	3	Nivel Dominio	Grupal	Redes Mundiales	Modelo Dominio
Intercambio Complejo de Medios	Distribuido	2	Nivel Programa	Automatización	Redes Locales	Modelo del Programa
Intercambio Electrónico Simple	Conectado	1	Nivel Sitio / Local	Manejadores del Sistema	Conexión Simple	Local
Entrada Manual	Aislado	0	Control de Acceso	N / A	Independiente	Privado

Figura 1.Modelo de madurez de la interoperabilidad.

1.2.2 Ejemplos de interoperabilidad

En víspera de que las comunicaciones de dos o más sistemas estén por establecerse deben seguir estándares de interfaces abiertos que existen, como son TCP/IP, HTTP y HTML que se muestran en la figura 2 en función del modelo OSI (*Open Systems Interconnection*), también se puede lograr la interoperabilidad por medio de un *middleware* que pueda ejercer de enlace entre diferentes aplicaciones. Ver figura 2.

Aplicación	Presentación	Sesión	Transporte	Red	Ligado de datos	Física
HTML	HTTP		TCP	IP		

Figura 2. Localización de estándares abiertos según el modelo OSI.

Ventajas

- Elude la dependencia de un solo fabricante, ya que si este decide retirar un determinado producto del mercado otro puede suplantarlo, por lo que la interoperabilidad posibilita proteger la continuidad del sistema en cuestión.
- Garantiza que los distintos fabricantes se ajusten a un estándar y así poder comunicarse.
- Exige a los fabricantes el desarrollo de soluciones técnico-económica superiores a las actuales, ya que si aparece una mejor solución y esta no está relacionada con los fabricantes actuales puede haber reemplazos, esto posibilita mejora continua.

- Permite transparencia en los procesos ya que es una misiva de que el fabricante apuesta por los estándares establecidos por la comunidad internacional y que el mismo va a competir limpiamente en un entorno de mercado que fomenta la calidad técnica y precios.
- Permite la reutilización de servicios ya que todos los sistemas conectados al *software* de interoperabilidad pueden consumir recursos de otros, incluso de nuevos sistemas que se incorporen.

(ALBENTIA s/f)

1.3 Sistemas Distribuidos

En función de definir un sistema distribuido primeramente se hablará de computación distribuida que es un término más general en donde se manifiesta un sistema en que muchos agentes autónomos, dotados con la capacidad de procesar información individualmente , intercambian datos entre sí, influyendo en el comportamiento de los mismos o sea la computación distribuida alude a los servicios que proporciona un sistema de computación distribuido, por lo que “un sistema distribuido se define como una colección de computadores autónomos conectados por una red, y con el *software* distribuido adecuado para que el sistema sea visto por los usuarios como una única entidad capaz de proporcionar facilidades de computación.” (2)

Un sistema distribuido que funcione como sistema deberá cumplir ciertas propiedades (Pech F. 2011, Chavez R. 2015 y Drake J. s/f) que se muestran a continuación:

Compartición de recursos

Para una compartición de recursos práctica, se debe poseer un *software* con una interfaz de comunicación que permita de forma segura la gestión de dichos recursos, en función de esto se puede decir que un sistema distribuido puede verse como un conjunto de gestores de recursos y un conjunto de programas que utilizan dichos recursos. Esto genera dos modelos, el modelo cliente servidor y el modelo basado en objetos.

Apertura

Un sistema informático puede ser abierto o cerrado en función del *hardware* o a las extensiones de *software*, por lo que la apertura en sistemas distribuidos está dada por el grado en que los nuevos servicios de compartición de recursos se puedan añadir sin duplicar o dañar los ya existentes.

Concurrencia

En los sistemas distribuidos convergen muchas máquinas, donde las cuales poseen de uno a varios procesadores centrales, o sea si hay N ordenadores con un procesador central cada uno, entonces se ejecutarán hasta N procesos paralelamente.

Tolerancia a fallos

Los sistemas distribuidos presentan gran disponibilidad ante los fallos, ya que cuando falla un componente en el sistema distribuidos solo se afectan las funciones relacionadas con ese componente y no a todo el sistema.

Transparencia

La transparencia se define como el encubrimiento al usuario y al programador de aplicaciones que separan los componentes de un sistema distribuido.

Ventajas

- **Disponibilidad y fiabilidad.** El sistema puede estar replicando los recursos de manera que, si una máquina pasa a una posición inactiva, el sistema como un todo puede seguir funcionando.
- **Bajo coste.** El sistema puede estar estructurado por computadoras sin grandes prestaciones.
- **Crecimiento.** Al sistema se le pueden añadir nuevos sistemas incrementando su poder de cómputo total.
- **Acceso a datos.** Ofrece facilidades de compartición de recursos y datos.
- **Transparencia**. El usuario está ajeno a la arquitectura que se emplee en el sistema.

(Pech F. 2011 y Taipe R. 2012)

Desventajas

- **Seguridad.** Gestión de seguridad compleja debido a que generalmente hay muchas empresas independientes.

- **Puntos de fallo.** Dado que gran cantidad de computadoras están relacionadas con el sistema distribuido y que estas necesitan de la red, pueden existir puntos en los cuales se presenten fallos en los enlaces de red o computadoras y en consecuencia presente problemas para el sistema distribuido como son la perdida de mensajes y latencia.
- **Gestión de recursos.** Si los recursos están mal distribuidos, algunos de estos pueden estar saturados mientras que otros en desuso.
- **Complejidad.** Los sistemas distribuidos son muy difíciles de diseñar.

(Pech F. 2011 y Taipe R. 2012)

1.4 Sistemas similares

1.4.1 CORBA

CORBA *(Common Object Request Broker Architecture*) es un estándar abierto de del OMG (*Object Management Group*) que favorece el desarrollo de aplicaciones distribuidas en ámbitos distribuidos. Está fundamentada en tres conceptos fundamentales que son el modelo orientado a objetos, entorno de computación distribuido y abierto, integración y reutilización de componentes. Además, especifica servidores estandarizados por medio de un modelo de referencia, los patrones de interacción entre clientes y servidores, y las especificaciones de las APIs (*Application Programming Interface*) favoreciendo el diseño de aplicaciones en plataformas heterogéneas sin necesidad de conocer los detalles de los recursos y servicios que ofrece cada elemento de la plataforma sin importar el lenguaje de programación, así como la interoperabilidad entre *software,* sujetos a diferentes tecnologías. A continuación, se muestra la arquitectura de CORBA en la figura 3. (Drake J. s/f y Millán R. 2016)

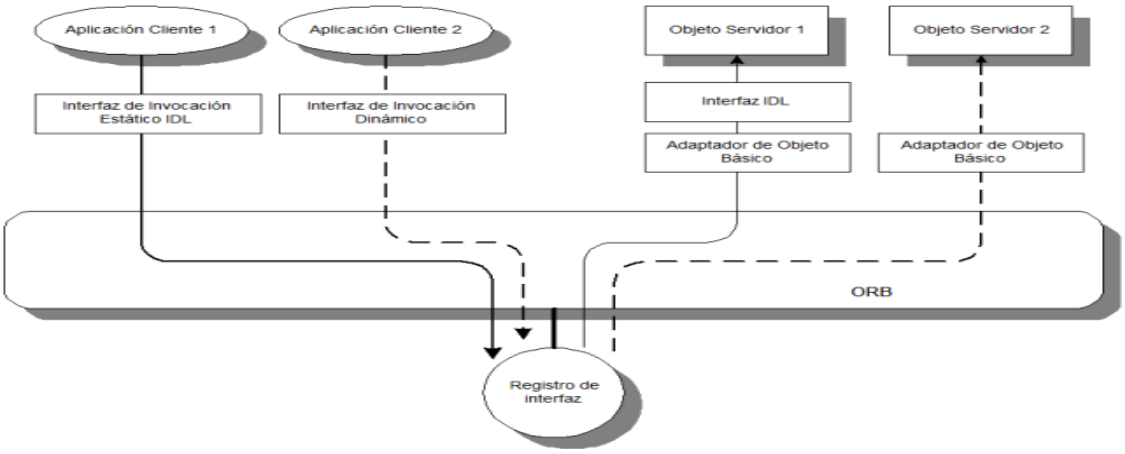


Figura 3. Arquitectura CORBA.

Ventajas

- Capacidad de acceso a objetos en cualquier subsistema de la plataforma.
- Permite la integración de aplicaciones que implementan diferentes tecnologías.
- Es seguro y sólido en lo que a transacciones se refiere.
- Provee muchas formas de invocaciones de los objetos que se manejan.
- Se puede programar en diversos lenguajes.

(Drake J. s/f)

Desventajas

- No es la tecnología más fácil de utilizar.
- Desarrollo costoso de especificaciones.

1.4.2 OPC UA

OPC (*OLE (Object Linking and Embedding) for Process Control*) unificada es un estándar de comunicación en el área de supervisión y control de desarrollo industrial que suministra una interfaz usual para la comunicación que posibilita que productos de software de diferentes proveedores compartan datos e interactúen entre sí. Es una arquitectura multiplataforma que funciona hasta en dispositivos inteligentes y controladores que ameriten sistemas de funcionamiento específico con capacidad de funcionamiento en tiempo real. Además, determina dos pilares primordiales en que se basa la interoperabilidad: la infraestructura de comunicaciones y el meta modelo OPC UA. A continuación, se muestra la arquitectura de OPC UA en la figura 4.

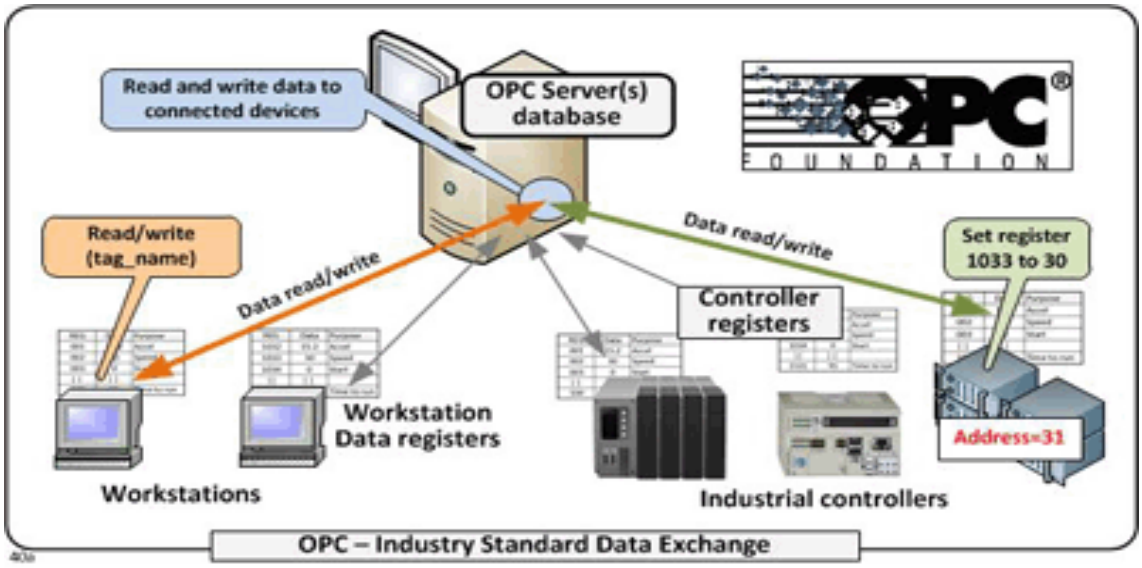


Figura 4. Arquitectura OPC UA

Características y beneficios de OPC UA

- Plataforma neutral que funciona en cualquier sistema operativo.
- Preparada para el futuro y para comunicar con sistemas antiguos.
- Fácil configuración y mantenimiento.
- Tecnología orientada a servicios.
- Aumento de la visibilidad.
- Mayor alcance de la conectividad.
- Alto rendimiento.

(OPC Unified Architecture s/f)

1.4.3 ZeroC ICE

ZeroC ICE es un *middleware* orientado a objetos en un marco RPC que mejora la productividad ya que no hay que tener en cuenta como se conecta a la red, así como la serialización y deserialización de los datos en función de su transmisión por la red. Además, hace invocaciones síncronas y asíncronas utilizando los protocolos TCP, UDP, SSL / TLS y tecnología WebSocket permitiendo a un servidor de reutilizar una conexión establecida por un cliente para hacer devoluciones de llamada. También en funciones de seguridad es potente ya que utiliza cifrado de datos y autenticación de conexiones mediante el *plugin* IceSSL. Es rápido, permitiendo que las aplicaciones atiendan a miles de clientes con facilidad mediante la utilización de un protocolo binario compacto que a su vez minimiza el consumo de ancho de banda. A continuación, se muestra la arquitectura de ZeroC ICE en la figura 5. (Spruiel M.2015 y ZeroC 2015)

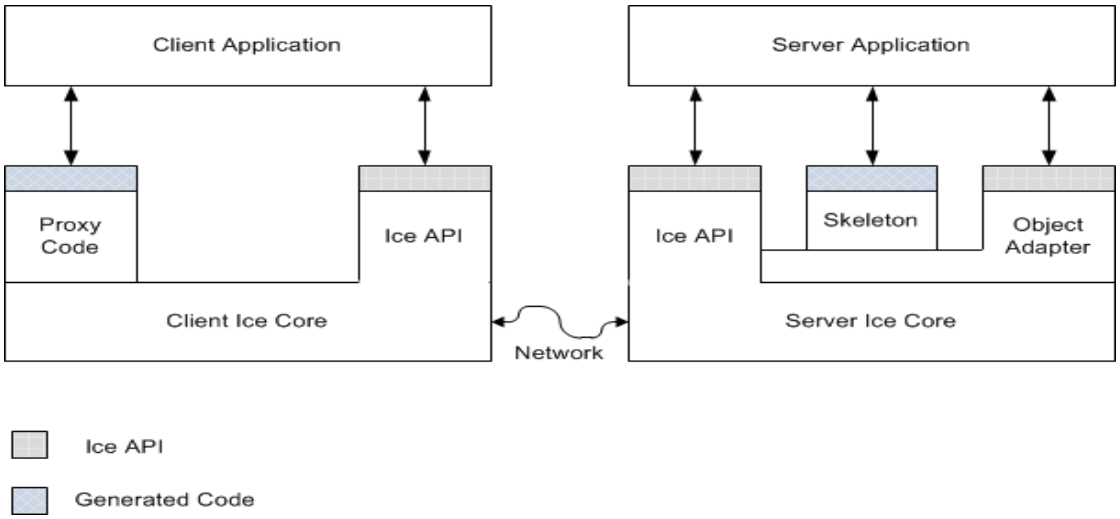


Figura 5. Arquitectura ZeroC Ice.

Beneficios arquitectónicos

- Semántica orientada a objetos.
- Invocación síncrona/asíncrona.
- Independencia (plataforma, sistema operativo, lenguaje, protocolo de transporte).
- Seguro.
- Multi-hilo.

No se adopta ninguna de estas tecnologías ya que son demaciado complejas para basadas en las mismas, desarrollar un módulo de interoperabilidad para el CMI en el SCADA Sainux.

1.5 Inteligencia de negocio

La inteligencia de negocio es el proceso a través del cual las empresas almacenan datos, los analizan y los transforman en conocimiento que utilizan para elaborar un plan o una estrategia comercial. Además, como estrategia empresarial incrementará la capacidad de tomar decisiones oportunas a la empresa, así como utilizar eficientemente sus recursos y supervisar sus objetivos.

¿Por qué inteligencia de negocios según?

- **Manejo de datos.** Tener datos de los clientes, empleados y subsistemas es importante, pero no es suficiente si se quiere ser competitivo. Para ser competitivo impera la necesidad de incrementar el conocimiento de los clientes y subsistemas mediante el análisis de los datos, lo cuales evidenciarán patrones de comportamiento, y la manera en que se monitoree, rastree y supervise este conocimiento permitirá a la empresa maximizar su rendimiento.
- **Fragmentación.** Poseen aplicaciones independientes por todos los subsistemas de la empresa, que ofrecen cada una, una versión de la realidad. Esto provoca que no se aprovechen eficientemente los datos.
- **Poca agilidad.** De acuerdo con el ineficiente manejo de datos y la fragmentación se necesita una herramienta que se ajuste a las necesidades de la empresa.

(ORACLE, s/f)

Ventajas

- **Control de costes.** Teniendo una herramienta que una los diferentes subsistemas de la empresa.
- **Entender mejor a los clientes.** Teniendo información de los clientes transformada en conocimiento basándose en patrones de comportamiento permite a la empresa retener clientes, ver nuevas oportunidades, efectividad de campañas.
- **Mejora la colaboración y calidad de las decisiones.** Teniendo una herramienta que posibilite la integración de todos los subsistemas.
- **Indicadores de gestión.** Permiten monitorear los procesos críticos de negocio, analizar el origen de los problemas y administrar los recursos y procesos en aras de trazar directrices para la toma de decisiones.
- **Asiste a los ejecutivos.** Provee a los ejecutivos de la empresa una panorámica general de los procesos de negocio que se llevan a cabo.

(Menéndez, J. s/f)

Desventajas:

- El incremento masivo de datos sin una adecuada gestión puede causar saturación en los almacenes de datos.
- Cuando las bases de datos no están normalizadas la extracción, transformación y carga de información pude resultar costosa por la existencia de datos duplicados, así como caracteres erróneos.

(Espinoza, G. y Galarza, D. 2012)

1.6 Cuadro de Mando Integral

El Cuadro de Mando Integral es una herramienta estratégica que posibilita establecer y supervisar los objetivos de la empresa y sus subsistemas mediante indicadores que muestran señales de alerta.

Principales objetivos del Cuadro de Mando Integral:

En la actualidad es evidente que las empresas desarrollan su ciclo de vida en entornos competitivos y globalizados en los cuales prima la necesidad de obtener resultados satisfactorios por lo que se impone gestionar recursos, procesos y acciones enfocadas en:

- Aclarar la visión y estrategia trazando directrices funcionales.
- Comprometer a todo el personal de la empresa a trabajar en aras de cumplir la estrategia definida.
- Integrar los objetivos e indicadores estratégicos.

(Páez F. s/f)

Las perspectivas del negocio encierran los objetivos estratégicos, indicadores, metas, además de proyectos estratégicos, en función de ofrecer una visión global de la empresa. Aunque estas perspectivas se pueden adaptar según las necesidades de la empresa, existen cuatro tradicionales definidas por Norton y Kaplan:

- La **“perspectiva financiera”** avala las expectativas y requisitos de los accionistas y establece métricas en función de la creación de valor de la empresa teniendo en cuenta el ciclo de vida de los servicios y productos.

- La **“perspectiva del cliente”** evidencia como se ilustra la empresa en el mercado, así como la manera en que se va a complacer los requisitos del cliente y que diferentes estrategias requieren desiguales proposiciones de valor.
- La **“perspectiva interna”** responde a la exigencia de enfocar indicadores de procesos internos de la empresa que son críticos para el posicionamiento en el mercado y para llevar la estrategia al mejor punto de satisfacción.
- La perspectiva de **“aprendizaje y crecimiento”** se centra en focalizar los esfuerzos en bienes materiales y humanos tales como promover la competencia en la empresa, así como potenciar la superación personal.

(Fernández A. s/f y SINNEXUS, 2016)

Beneficios de la implantación de un Cuadro de Mando Integral:

- Funciona como intérprete del modelo de negocio y provee facilidades en las concesiones de la directiva de acuerdo a como lograr los objetivos deseados.
- Contribuye a un mejor control de los objetivos trazados a corto, mediano y largo plazo en función de la estrategia elaborada, permitiendo tomar decisiones de manera oportuna.
- Monitorea las relaciones causa-efecto de la estrategia, así como las desviaciones en el plan estratégico u operativo posibilitando actuar de forma proactiva ante las posibles oportunidades o debilidades identificadas.

(SINNEXUS, 2016)

Riesgos de la implantación de un Cuadro de Mando Integral:

- Debe tener información relevante, actual y fiable sino puede no aprovecharse las facilidades que provee el CMI para la detección de debilidades y oportunidades, y convertirse en enemigo de lo que realmente desea la empresa.
- Si el CMI no actúa sobre los puntos críticos de la gestión, se pierde valor de uso, en función al mensaje que se desea emitir.
- La no inclusión de la visión de la dirección de la empresa en la elaboración del modelo a seguir traerá un enfoque no deseado y esfuerzo mal gestionado.

(SINNEXUS, 2016)

1.7 Tecnologías a utilizar

En el desarrollo de un proyecto la selección de la tecnología a utilizar marca el grado de rendimiento, calidad y modernidad de dicho proyecto. En función de dicha selección se debe tener en cuenta la disponibilidad de las herramientas, curva de aprendizaje, licencias y objetividad.

1.7.1 Node.js

"Node.js es una plataforma construida encima del entorno de ejecución JavaScript de Chrome para fácilmente construir rápidas y escalables aplicaciones de red. Node.js usa un modelo de E/S no bloqueante dirigido por eventos que lo hace ligero y eficiente, perfecto para aplicaciones *data-intensive* en tiempo real" (3). Node soporta e implementa el sistema que CommonJS define para esta gestión de módulos, lo cual es vital a la hora de crear un ambiente de variables dentro de los módulos que no afecte a variables del mismo nombre en un ámbito global proporcionando ventajas de acuerdo a reutilización de componentes de proyectos (Criteria Studio, 2012). Esta tecnología permitió que se construyera un módulo de interoperabilidad porque esta proporciona una solución adecuada para aplicaciones en tiempo real por su bajo consumo de recursos y rapidez. Además, tiene una comunidad muy activa y es fácil de aprender.

Las implicaciones del uso de este estándar son:

- La función *require* () permite la utilización de bibliotecas desarrolladas para Node.js, tales como http, ws, mongodb, destinadas para crear servidores y conexiones a base de datos respectivamente.
- La variable *exports* dentro de los módulos actúa como un objeto que posibilita el control de acceso a las variables de dichos módulos.
- La variable *module* dentro de un módulo provee a este de una instancia única, identificando a este de los restantes módulos, y a través de esta se accede a la variable *exports* que se necesite.

(Muñoz A. 2013)

Diferencias que tiene Node.js respecto a Apache u otros servidores web

- Apache funciona adecuadamente para pocas conexiones ya que el mismo maneja un hilo por cada conexión y a medida que estas incrementan, el consumo de recursos del sistema aumenta considerablemente.

- Node.js posee la virtud de mantener muchas conexiones abiertas y esperando ya que consiste en un hilo principal que atiende a los usuarios del sistema y otro conjunto de hilos para las operaciones E/S de forma asíncrona, lo que permite aprovechar mejor los recursos del sistema. Hipotéticamente Node.js puede mantener tantas conexiones como número máximo de archivos descriptores (sockets) soportados por el sistema y si este es UNIX puede rondar las 65.000 conexiones. Sin embargo, Apache después de manejar una cantidad de clientes superior a 256 corre el riesgo de bloquearse, todo esto en función de los recursos de las computadoras.
- Debido a que Node.js en función de su arquitectura trabaja sobre un solo hilo solo se podrá usar un CPU y esto puede convertirse en una desventaja si no se inician múltiples instancias del mismo en el servidor y se pone un manejador de fluctuación de carga.

Node.js trabaja con una herramienta conocida como Express que es un "*framework* de desarrollo de aplicaciones web minimalista y flexible" (3). Está basado en *Connect* que es un *framework* extensible de manejo de servidores HTTP. Además, es flexible, rápido, muy simple.

Características de Express

- Robusto sistema de enrutamiento de peticiones.
- Soporte para generar páginas HTML dinámicas a partir de plantillas con capacidad de utilizar varios motores de renderizado de vistas.
- Ofrece router de URL (Get, Post, Put, Delete, Update).
- Facilidad para motores de plantillas (Jade, EJS, JinJS.).

Ventajas de Node.js

- Maneja un alto grado de operaciones de Entrada/Salida de manera eficiente.
- Es óptimo como capa superior de fuentes de datos como bases de datos u otros servicios web.
- Es perfecto para aplicaciones en tiempo real.
- Presenta una comunidad enorme y muy activa.
- Posibilita gestionar en el servidor accesos a ficheros, conexiones a clientes, base de datos, etc.

(Muñoz A. 2013)

1.7.2 WebSocket

WebSocket es una tecnología que provee una conexión full dúplex (ida y vuelta de forma simultanea) entre el cliente y servidor respectivamente en tiempo real bajo el protocolo TCP, como se muestra en la figura 6. La latencia en el intercambio de datos es mínima ya que el socket está siempre abierto y escuchando comparada con el paradigma AJAX, en la cual hay ejecutar una petición, procesarla y enviar una respuesta. Esta tecnología posibilitó al módulo de interoperabilidad la integración con los demás módulos del CMI y tiene bibliotecas en Node.js que permite el desarrollo de aplicaciones basadas en la misma. (Grigorik I. 2013 y Lombardi A. 2015)

Principios

- Se inicia con un *handshake* (proceso automatizado mediante el cual dos entidades atreves de un canal de comunicación establecen parámetros para comunicarse) HTTP, por lo que si no hay HTTP no hay WebSocket.
- Tanto servidor como cliente tienen que soportarlo.
- La conexión TCP va por el puerto 80.

(Grigorik I. 2013 y Lombardi A. 2015)

Características

- Atraviesa *proxies*, *routers* y *firewalls*.
- Solo se puede enviar texto en JSON que es un formato de texto ligero compuesto por una colección de pares nombre/valor o una lista ordenada de valores y el mismo es independiente del lenguaje lo que lo hace excelente para el intercambio de datos.
- Comparte el puerto con HTTP.
- La conexión se establece mediante el *upgrade* del protocolo HTTP al protocolo Websocket usando la misma conexión.

(Grigorik I. 2013 y Lombardi A. 2015)

Ventajas

- Es más óptimo que el protocolo HTTP ya que este contiene mucha más información a lo que cabecera de datos se refiere que WebSocket, por lo que consume más recursos.

- Debido a que WebSocket consume menos recursos en el servidor, reduce la latencia en las conexiones e incrementa el número de conexiones entre el cliente y servidor.
- La transmisión de información es más rápida que el protocolo HTTP.
- Favorece la escalabilidad en la red ya que es un protocolo que a la hora de establecer una conexión persistente con el servidor lo hace de manera destacada.

(Chavez R. 2015)

Desventajas

- Es una tecnología joven y en continuo desarrollo, por lo que se pueden presentar problemas en la conexión con algunos navegadores.
- Falta de fiabilidad ya que se encontraron vulnerabilidades en 2010 lo que desencadenó que muchos navegadores incapacitaran sus funciones.

(Chavez R. 2015)

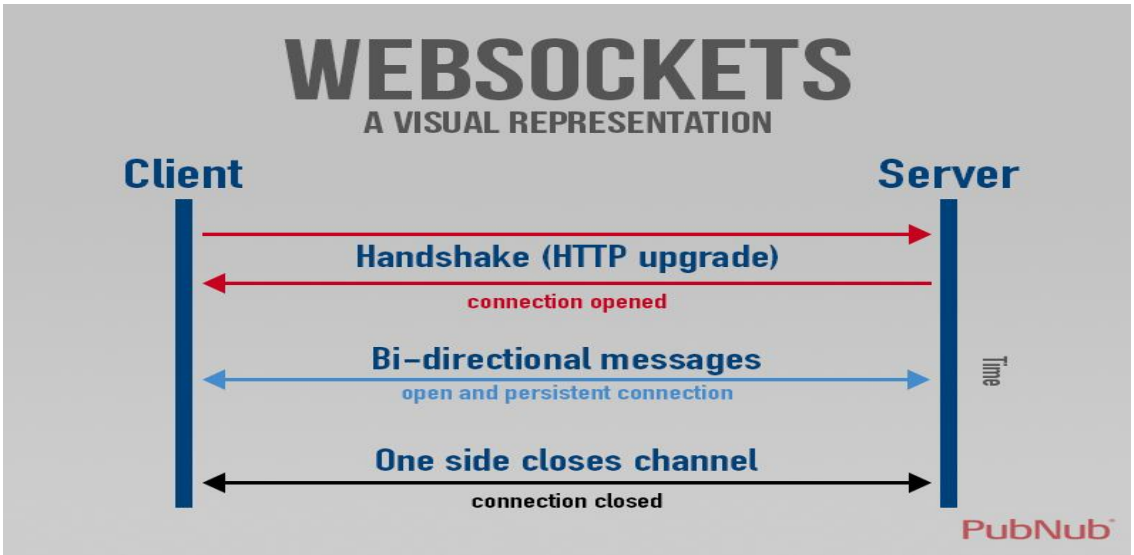


Figura 6. Funcionamiento de WebSocket.

1.7.3 MongoDB

MongoDB es una base de datos NoSQL orientada a documentos, o sea los datos se guardan en documentos en vez de registros como en las bases de datos tradicionales, donde el formato de guardado se especifica como BSON que es una representación binaria de JSON. Para el manejo de datos no es necesario seguir un esquema, lo cual justifica que los documentos de una misma colección (concepto similar a una base de datos relacional), puedan tener estructuras diferentes potenciando una enorme flexibilidad. Esta tecnología permitió al módulo de interoperabilidad asegurar la persistencia de los datos relacionados a los mensajes que no se pudieron mandar a un determinado destinatario ya que este estaba fuera de servicio, así como el acceso rápido y sencillo a estos datos para su posterior procesamiento. Además, se considera su uso porque también provee replicación de sus bases de datos y en situaciones de desastres esto puede resultar en la no perdida de información. También presenta bibliotecas en Node.js para desarrollar aplicaciones que requieran el manejo de datos y como la misma está basada en JavaScript al igual que Node.js hace más fácil su utilización. (Alarcón J.2014 y Seguin K. 2014)

Principales características:

- Está fundamentado en el motor V8 de Google Chrome para JavaScript lo cual hace fácil su aprendizaje.
- Almacenamiento basado en JSON sin necesidad de definir esquemas previamente lo que lo hace dinámico y flexible.
- Alto rendimiento para consultas y actualizaciones.
- Alta capacidad de replicación, crecimiento y escalabilidad ya que puedes escalar horizontalmente añadiendo máquinas a bajo costo sin ver afectado el rendimiento ni presentar problemas con la gestión.

(Alarcón J. 2014 y Seguin K. 2014)

Ventajas

- **Escalabilidad:** Posibilitan ir aumentando el poder de procesamiento a medida que los requerimientos de datos suben.
- **Flexibilidad:** No presentan un esquema preestablecido, por lo que se puede gestionando la configuración de los datos del *software* sin alterar los ya definidos.

- **Sencillez:** Los motores NoSQL generalmente no dependen de un administrador de base de datos, lo que los hace más prácticos.

(Alarcón J. 2014 y Seguin K. 2014)

Desventajas

- **Falta de madurez:** Se considera una tecnología joven por el tiempo que lleva en el mercado, lo que no genera confianza para los que quieran realizar aplicaciones más complejas.

1.8 Lenguajes de programación

1.8.1 JavaScript

JavaScript es un dialecto de la especificación estándar ECMA-262, el cual se acopla perfectamente al paradigma de la programación orientada a eventos,en la que el flujo del *software* no sigue una secuencia de pasos predeterminada sino que depende de determinados eventos que se accionan de manera asíncrona generalmente, durante la ejecución de dicho *software*. Se empleó este lenguaje ya que Node.js esta basado en el mismo. (Muñoz A. 2013)

Características

- **Funcional.** La programación funcional es un paradigma de progrmación que posibilita por ejemplo la transpaencia referencial que establece que para un valor de entrada siempre se produce la misma salida, esta transparencia provee a su ves facilidades para las pruebas y depuración.
- **Orientado a objetos parcialmente.** Presenta herencia por prototipado y la encapsulación mediante funciones dentro de funciones, pero no soporta polimorfismo.
- **Debilmente tipado.** En la declaración de variables no se exige la relación con un tipo de dato determinado.
- **Funciones anónimas.** Son muy convenientes en los llamados *callback* (devolución de llamada) para el manejo de determinado evento o sea cuando un proceso termina avisa para así ejecutar la función siguiente.

(Muñoz, A. 2013)

Ventajas

- Es un lenguaje multiplataforma.
- Es facil de aprender.

1.8.2 Java

Java es un lenguaje de programación orientado a objeto de alto nivel que se caracteriza por ser simple o sea tiene una curva de aprendizaje alta, interpretado, seguro, multiplataforma y multi-hilo que es esencial a la hora de desarrollar aplicaciones de red distribuidas ya que un hilo puede atender a la comunicación mientras otros pueden interactuar con las bases de datos o usuarios. Posee una amplia documentación accesible a la comunidad de manera gratuita y excelente cantidad de APIs disponibles que contribuyen a un mejor desempeño del programador. Se utilizó este lenguaje en el desarrollo de la fachada para la comunicación con el módulo del CMI relacionado a la recuperación de información ya que el mismo utiliza este lenguaje.

Ventajas

- Robusto.
- Multitarea.
- Dinámico.

1.9 Entornos de desarrollo

1.9.1 Brackets

Brackets es un editor de código abierto para el desarrollo y diseño web que soporta lenguajes como HTML, CSS y JavaScript. Es integrable con el sistema de control de versiones Git e incorpora conexión en tiempo real con el navegador. Se utilizó en el desarrollo de la fachada del módulo de visualización del CMI y el módulo de interoperabilidad. (Guaita A. 2012)

Características

- Está enfocado para el desarrollo web.
- Soporta tres tipos de lenguaje (html, css y javascript).
- Se le pueden incorporar *plugins* para gestionar aplicaciones basadas en Node.js.

(Guaita A. 2012)

1.9.2 NetBeans

Netbeans es un Entorno de Desarrollo Integrado (IDE) que facilita el desarrollo de aplicaciones de escritorio y web permitiendo que dichas aplicaciones se construyan a partir de conjuntos de componentes de *software* llamados módulos.Una aplicación desarrollada en dicho IDE puede contener x cantidad de módulos donde cada uno responde por determinadas responsabilidades y constribuye a la eficiencia del programador. Se empleó en el desarrollo de la fachada del módulo de recuperación de información.

Características

- Es multiplataforma y multilenguaje podiendo soportar lenguajes como Java,PHP y C++.
- Cuenta con una comunidad de usuarios activa y con una vasta cantidad de documentación.
- Posibilita la inclusión de módulos que posibilita incrementar las funcionalidades.

1.10 Lenguajes de modelado

1.10.1 Lenguaje Unificado de Modelado

El Lenguaje Unificado de Modelado (UML) es un lenguaje basado en un vocabulario y reglas que permite modelar gráficamente los sistemas en cuestión y es independiente de los métodos de análisis y diseño. Modela sistemas tales como sistemas de *software*, sistemas de *hardware*, y organizaciones del mundo real. Se utilizó para modelar el CMI y el módulo de interoperabilidad en función de un mejor entendimiento del sistema. (Larman C. 2003 y Orallo E. s/f)

Objetivos

- **Visualizar.** Posibilita reflejar de forma gráfica un sistema estableciendo las relaciones que existen en este, sus componentes, puertos que utiliza entre otras cuestiones tales como clases.
- **Especificar.** Permite modelar las características concretas de un sistema determinado.
- **Simple.** Ser simple sin perder la capacidad de modelar cualquier sistema.
- **Construir.** En función de determinadas especificaciones construir sistemas.

- **Independiente.** Es independiente del lenguaje de programación y del proceso que se lleve a cabo en la construcción de determinado *software*.

Ventajas

- Facilita la gestión de la aplicación a desarrollar.
- Propicia la reutilización.
- Contribuye grandemente a el control de proyectos.
- Modela sistemas tanto informáticos como del mundo real.

(Orallo E. s/f)

Desventajas

- No se puede establecer la duración de las actividades.
- Cuando los diagramas son muy grandes su comprensión se dificulta.

1.11 Metodologías de desarrollo

1.11.1 AUP-UCI

La UCI adoptó la posición de variar la metodología Proceso Unificado Ágil (AUP) en función de que esta se adapte al ciclo de vida definido para la actividad productiva de la misma, basándose en el Modelo CMMI-DEV v1.3 que establece directrices que fomentan excelentes prácticas para desarrollar productos y servicios de calidad.

Fases

De las 4 fases que propone AUP (Inicio, Elaboración, Construcción, Transición) se decide para el ciclo de vida de los proyectos de la UCI mantener la fase de Inicio, pero modificando el objetivo de la misma, se unifican las restantes 3 fases de AUP en una sola, a la que se llamará Ejecución y se agrega la fase de Cierre. De las 4 fases que propone AUP (Inicio, Elaboración, Construcción, Transición) se decide para el ciclo de vida de los proyectos de la UCI mantener la fase de Inicio, pero modificando el objetivo de la misma, se unifican las restantes 3 fases de AUP en una sola, a la que se llamará Ejecución y se agrega la fase de Cierre. Para una mayor comprensión se muestra la siguiente Tabla 1:

Fases Variación AUP-UCI	Objetivos de las fases (Variación AUP-UCI)
Inicio	Durante el inicio del proyecto se llevan a cabo las actividades relacionadas con la

	planeación del proyecto. En esta fase se realiza un estudio inicial de la organización cliente, que permite obtener información fundamental acerca del alcance del proyecto, realizar estimaciones de tiempo, esfuerzo y costo y decidir si se ejecuta o no el proyecto.
Ejecución	En esta fase se ejecutan las actividades requeridas para desarrollar el software, incluyendo el ajuste de los planes del proyecto considerando los requisitos y la arquitectura. Durante el desarrollo se modela el negocio, obtienen los requisitos, se elaboran la arquitectura y el diseño, se implementa y se libera el producto. Durante esta fase el producto es transferido al ambiente de los usuarios finales o entregado al cliente. Además, en la transición se capacita a los usuarios finales sobre la utilización del software.
Cierre	En esta fase se analizan tanto los resultados del proyecto como su ejecución y se realizan las actividades formales de cierre del proyecto.

Tabla 1. Fases AUP-UCI.

Descripción de las disciplinas

AUP propone 7 disciplinas (Modelo, Implementación, Prueba, Despliegue, Gestión de configuración, Gestión de proyecto y Entorno), se decide para el ciclo de vida de los proyectos de la UCI tener 8 disciplinas, pero a un nivel más atómico que el definido en AUP. Los flujos de trabajos: Modelado de negocio, Requisitos y Análisis y diseño en AUP están unidos en la disciplina Modelo, en la variación para la UCI se consideran a cada uno de ellos disciplinas. Se mantiene la disciplina Implementación, en el caso de prueba se desagrega en 3 disciplinas: Pruebas Internas, de Liberación y Aceptación y la disciplina Despliegue se considera opcional. Las restantes 3 disciplinas de AUP asociadas a la parte de gestión para la variación UCI se cubren con las áreas de procesos que define CMMI-DEV v1.3 para el nivel 2, serían CM (Gestión de la

configuración), PP (Planeación de proyecto) y PMC (Monitoreo y control de proyecto). Para una mayor comprensión se muestra la siguiente Tabla 2:

Disciplinas Variación AUP-UCI	Objetivos Disciplinas (Variación AUP-UCI)
Modelado de negocio (opcional)	El Modelado del Negocio es la disciplina destinada a comprender los procesos de negocio de una organización. Se comprende cómo funciona el negocio que se desea informatizar para tener garantías de que el software desarrollado va a cumplir su propósito.
Requisitos	El esfuerzo principal en la disciplina Requisitos es desarrollar un modelo del sistema que se va a construir. Esta disciplina comprende la administración y gestión de los requisitos funcionales y no funcionales del producto.
Análisis y diseño	En esta disciplina, si se considera necesario, los requisitos pueden ser refinados y estructurados para conseguir una comprensión más precisa de estos, y una descripción que sea fácil de mantener y ayude a la estructuración del sistema (incluyendo su arquitectura). Además, en esta disciplina se modela el sistema y su forma (incluida su arquitectura) para que soporte todos los requisitos, incluyendo los requisitos no funcionales. Los modelos desarrollados son más formales y específicos que el de análisis.
Implementación	En la implementación, a partir de los resultados del Análisis y Diseño se construye el sistema.
Pruebas interna	En esta disciplina se verifica el resultado de la implementación probando cada

	construcción, incluyendo tanto las construcciones internas como intermedias, así como las versiones finales a ser liberadas. Se deben desarrollar artefactos de prueba como: diseños de casos de prueba, listas de chequeo y de ser posible componentes de prueba ejecutables para automatizar las pruebas.
Pruebas de liberación	Pruebas diseñadas y ejecutadas por una entidad certificadora de la calidad externa, a todos los entregables de los proyectos antes de ser entregados al cliente para su aceptación.
Pruebas de Aceptación	Es la prueba final antes de despliegue del sistema. Su objetivo es verificar que el software está listo y que puede ser usado por usuarios finales para ejecutar aquellas funciones y tareas para las cuales el software fue construido.
Despliegue (Opcional)	Constituye la instalación, configuración, adecuación, puesta en marcha de soluciones informáticas y entrenamiento al personal del cliente.

Tabla 2. Disciplinas AUP-UCI.

Roles AUP-UCI

AUP propone 9 roles (Administrador de proyecto, Ingeniero de procesos, Desarrollador, Administrador de BD, Modelador ágil, Administrador de la configuración, *Stakeholder*, Administrador de pruebas, Probador), se decide para el ciclo de vida de los proyectos de la UCI tener 11 roles, manteniendo algunos de los propuestos por AUP y unificando o agregando otros. Para una mayor comprensión se muestra la siguiente Tabla.

- Jefe de proyecto.
- Planificador.
- Analista.
- Desarrollador.

- Administrador de la Configuración.
- *Stakeholder* (Inversor, accionista, etc).
- Administrador de calidad.
- Probador.
- Arquitecto de software.

Conclusiones del capítulo

Se realizó un estudio detallado y profundo de la interoperabilidad, así como las tecnologías que la posibilitan, arquitectura y metodología que se utilizará para la construcción de la aplicación. Se asume la tecnología Node.js como servidor que provee la comunicación mediante WebSocket, se emplea JavaScript como lenguaje de programación en el servidor y en la fachada del Módulo de visualización para el CMI del SCADA SAINUX, así como el lenguaje de programación Java para la facha del Módulo de Recuperación de Información de Bases de Datos para el CMI del SCADA-SAINUX, además como base de datos se asume MongoDB y para el desarrollo de las funcionalidades, el editor de texto Brackets y Netbeans como entorno de desarrollo. Además, se utilizará la metodología de AUP-UCI para contrarrestar los diferentes cambios que puedan ocurrir en el desarrollo del sistema.

Capitulo II: Descripción de la propuesta de solución

2.1 Introducción

En el presente capítulo se expondrá la propuesta de solución al problema existente, basado en la metodología AUP-UCI para la planificación, investigación y diseño del módulo. Se presentará una descripción del modelo de dominio, diagrama de clases correspondiente al modelo de objetos, así como requisitos funcionales y no funcionales del sistema en función de la realización de las historias de usuario. También se abordarán los patrones empleados durante el desarrollo y el diagrama de paquetes.

2.2 Modelo de dominio

El modelo de dominio es “la representación visual de los conceptos u objetos más importantes de un negocio, sus características y las relaciones entre dichos conceptos” (4) y se realiza con el objetivo de comprender y describir las clases más importantes dentro del contexto del sistema y, por tanto, también contribuir a la comprensión de los requisitos de dicho sistema, que se desprenden de este contexto. Ver figura 7.

2.2.1 Diagrama de clases de dominio.

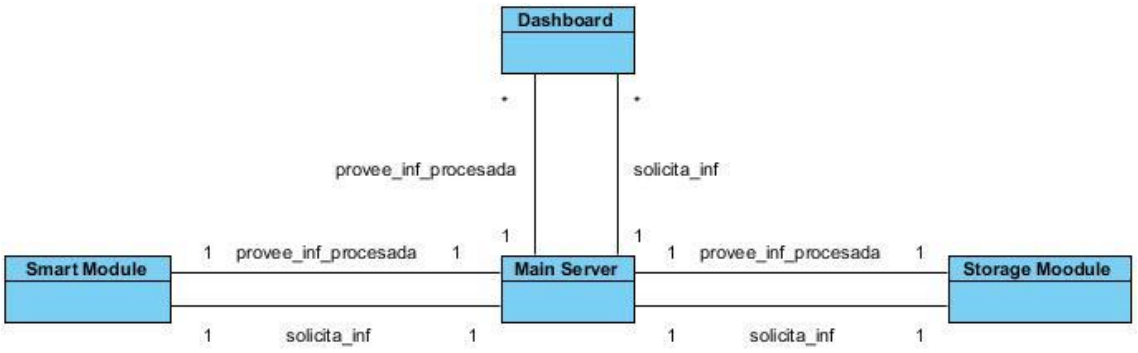


Figura 7. Diagrama de dominio.

2.2.2 Descripción del modelo de dominio

El módulo *Main Server* del CMI que se desarrolla, actúa de distribuidor de información en forma de mensajes entre los módulos *Smart, Storage* y *Dashboard,* donde a partir del subsistema que solicita información el *Server* tendrá la capacidad de conocer en que otro subsistema se encuentra dicha información, hacer la petición y generar una respuesta al subsistema que pretendió en un inicio la información.

2.3 Especificación de requisitos

En función del sistema a desarrollar se definen los requerimientos del mismo en aras de su implementación, en base a lo planteado se acotarán el número de variaciones que habrá que realizar en el producto debido a alteraciones en sus requisitos. Fundado en el modelo de dominio presentado se realizó el levantamiento de requisitos, donde los mismos se fraccionan en dos tipos esencialmente, los funcionales y los no funcionales. A continuación, se describen los requisitos puntualizados para el desarrollo del sistema propuesto. (Pressman R. 2001)

2.3.1 Requisitos funcionales

Los requisitos funcionales (RF) son servicios que el sistema debe cumplir, dependiendo del *software* que se desea desarrollar, así como la manera de enfocar las necesidades del usuario, sin tomar en consideración ningún tipo de restricción física. Además, evidencian el comportamiento del *software* en su entorno, así como las variaciones de este y su funcionamiento. A continuación, se muestran los requisitos funcionales de la solución propuesta. (Pressman R. 2001)

- **RF1 El sistema debe permitir aceptar conexión.**
- **RF2 El sistema debe permitir recibir mensajes de solicitud.**
- **RF3 El sistema debe permitir recibir mensajes de respuesta.**
- **RF4 El sistema debe permitir almacenar mensajes recibidos.**
- **RF5 El sistema debe permitir re-enviar mensajes almacenados.**

Las descripciones de cada uno de estos requisitos se encuentran en las historias de usuario correspondientes.

2.3.2 Requisitos no funcionales

Los requisitos no funcionales (RNF) son las propiedades o cualidades emergentes que el sistema debe tener, tales como la fiabilidad, el tiempo de respuesta, las necesidades de almacenamiento, entre otros que se abordarán a continuación. (Pressman R. 2001)

Fiabilidad.

- **RNF1:** Debe mantenerse la consistencia de los datos en correspondencia con la realidad.

Eficiencia.

- **RNF2:** El tiempo de respuesta estará dado por la cantidad de información a procesar, entre mayor cantidad de información mayor será el tiempo de procesamiento.
- **RNF3:** Al igual que el tiempo de respuesta, la velocidad de procesamiento de la información, la actualización y la recuperación dependerán de la cantidad de información que tenga que procesar.

Restricciones de diseño e implementación.

- **RNF4:** El software se desarrollará utilizando la metodología de desarrollo AUP-UCI.

Software.

- **RNF5:** Sistema Operativo Linux (en cualquier distribución) o Windows (7 o superior).

Requerimiento de hardware.

- **RF6:** Mínimo 4GB RAM, procesador Intel Dual Core o AMD A4 (1.9 GHz), HDD 120 GB.

2.4 Descripción del sistema propuesto

El sistema propuesto será un módulo de interoperabilidad que proveerá comunicación entre los diferentes subsistemas del CMI, el cual gestionará la información que soliciten los mismos. Para lograr una mejor comprensión de sistema propuesto se realizará una descripción de las historias de usuario en las tablas 3,4,5,6 y 7.

2.4.1 Historia de usuario para los requisitos funcionales de la aplicación

Historia de Usuario	
Número: 1	**Nombre de Historia de Usuario:** El sistema debe permitir aceptar conexión.
Programador: Felix Alejandro Paez Garriga	**Iteración Asignada:** 1
Prioridad en Negocio: Alta	**Puntos Estimados:** 2
Riesgo en Desarrollo: Alto	**Puntos Reales:** 3

Descripción: El sistema debe ser capaz de aceptar conexiones mediante el protocolo websocket. Identifica la conexión y almacena los datos referentes a la instancia. Se le notifica al cliente de que la conexión está establecida.
Observaciones: Los módulos deben tener una instancia de websocket.

Tabla 3. Historia de usuario 1.

Historia de Usuario	
Número: 2	**Nombre de Historia de Usuario:** El sistema debe permitir recibir mensajes de solicitud.
Programador: Felix Alejandro Paez Garriga	**Iteración Asignada:** 1
Prioridad en Negocio: Alta	**Puntos Estimados:** 2
Riesgo en Desarrollo: Alto	**Puntos Reales:** 3
Descripción: El sistema debe ser capaz de recibir mensajes en formato JSON, provenientes de una de las conexiones activas. Al recibir el mensaje de solicitud se identifica el origen del mismo mediante una propiedad "*key*" que tiene un valor que va a ser único por cada conexión activa. Luego se identifica el responsable de responder la solicitud del mensaje mediante una propiedad "*destiny*" que tiene un valor que va a ser único por cada conexión. Después se comprobará que el responsable de responder la solicitud tiene una conexión activa, entonces re-enviar el mensaje recibido. En caso de que el destinatario del mensaje-solicitud recibido no esté conectado, se almacena la solicitud y se re-envía en el momento en que este se conecte.	
Observaciones: Los módulos deben tener una instancia de websocket.	

Tabla 4. Historia de usuario 2.

Historia de Usuario	
Número: 3	**Nombre de Historia de Usuario:** El sistema debe permitir recibir mensajes de respuesta.
Programador: Felix Alejandro Paez Garriga	**Iteración Asignada:** 1
Prioridad en Negocio: Alta	**Puntos Estimados:** 2
Riesgo en Desarrollo: Alto	**Puntos Reales:** 3
Descripción: El sistema debe ser capaz de recibir respuesta a los mensajes re-enviados en formato JSON, provenientes de una de las conexiones activas. Al recibir el mensaje de respuesta se identificará el origen del mismo mediante una propiedad "*key*" que tiene un valor que va a ser único por cada conexión activa. Luego se identificará el quién realizó la solicitud de dicha respuesta. Después se comprobará que el responsable de la solicitud de respuesta tiene una conexión activa, entonces re-enviar el mensaje recibido. En caso de que el destinatario del mensaje de solicitud de respuesta no esté conectado, almacenar la solicitud y re-enviar en el momento en que este se conecte.	
Observaciones: Los módulos deben tener una instancia de websocket.	

Tabla 5. Historia de usuario 3.

Historia de Usuario	
Número: 4	**Nombre de Historia de Usuario**: El sistema debe permitir almacenar mensajes recibidos.
Programador: Felix Alejandro Paez Garriga	**Iteración Asignada:** 1
Prioridad en Negocio: Alta	**Puntos Estimados:** 2
Riesgo en Desarrollo: Alto	**Puntos Reales:** 3
Descripción: El sistema debe ser capaz de almacenar los mensajes que no se pueden enviar debido a que el destinatario de una solicitud o de una respuesta no esté conectado en el momento requerido.	
Observaciones: La base de datos mongoDB debe estar instalada y bien configurada.	

Tabla 6. Historia de usuario 4.

Historia de Usuario	
Número: 5	**Nombre de Historia de Usuario**: El sistema debe permitir re-enviar mensajes almacenados.
Programador: Felix Alejandro Paez Garriga	**Iteración Asignada:** 1
Prioridad en Negocio: Alta	**Puntos Estimados:** 2
Riesgo en Desarrollo: Alto	**Puntos Reales:** 3
Descripción: El sistema debe ser capaz en cuanto un subsistema se conecte reciba los mensajes almacenados dirigidos al mismo, en caso de poseerlos.	
Observaciones: La base de datos mongoDB debe estar instalada y bien configurada.	

Tabla 7. Historia de usuario 5.

2.5 Arquitectura de software

La arquitectura de *software* es de vital importancia ya que la forma en que se define un sistema establece un alto grado de incidencia en la capacidad del mismo para satisfacer los atributos de calidad del sistema. Permite la comunicación entre las partes involucradas en el desarrollo de un sistema. Además, incluye decisiones fundamentales relativas al diseño de aplicaciones, las cuales impactan el trabajo de ingeniería de software e influyen en el éxito final.

2.5.1 Patrón de arquitectura del software

En aras de desarrollar el módulo propuesto el autor considera utilizar el patrón de arquitectura del software de tres capas (Ver figura 8), donde las capas “son agrupaciones horizontales lógicas de componentes de software que forman la aplicación” (5) y estas contribuyen a enmarcar y separar las tareas a ejecutarse por dichos componentes dentro del sistema, fomentando el mantenimiento del código y la reutilización, lográndose una alta cohesión y bajo acoplamiento.(Torre C. y otros 2010)

Figura 8. Patrón de arquitectura del software de tres capas.

Se asume “un diseño en Capas estricto”, en función de que los componentes de una capa puedan interactuar con los componentes de su misma capa o de nivel inmediatamente inferior, por lo que la capa de comunicación interactuará con la capa de control y esta a su vez con la capa de persistencia. (Torre C. y otros 2010)

Las tres capas mencionadas anteriormente se describen a continuación con el objetivo de mejorar su comprensión.

Capa de conexión

En esta capa se manejan las principales bibliotecas que proveen la comunicación tales como http y ws (biblioteca para WebSocket), así como la lista de usuarios que están conectados. Además, se maneja una instancia de la capa control y mediante esta se transmite fundamentalmente, la lista de usuarios conectados y la instancia de WebSocket referente al usuario que está consumiendo o pretenda consumir los recursos del sistema, para su gestión.

Capa de control

Esta capa suministra un control referente a la estructura del mensaje JSON enviado por cualquier usuario. Además, provee un sistema de eventos que de acuerdo a la información que se solicite se ejecutarán, siendo los fundamentales *login* y *message,* donde el primero permite referenciar al usuario en el sistema y el segundo a gestionar la información referente a sus intereses. También se maneja una instancia de la capa de persistencia y mediante esta se transmite la información para, o guardar el mensaje JSON debido a que un usuario no presente una conexión activa o acceder a los

mensajes guardados (en caso de poseerlos) referentes al usuario en cuestión cuando se conecte al sistema.

Capa de persistencia

En esta capa se administran las principales bibliotecas referentes a la base de datos y las interfaces de acceso a la misma, permitiendo que la información que no se pueda mandar debido a que un usuario no esté conectado se guarde de manera satisfactoria. También posibilita el acceso a la información guardada referente a un usuario y su envío (en caso de poseerla) en cuanto este se conecte al sistema.

2.6 Patrones de diseño

Los patrones de diseño posibilitan resolver diferentes problemas en el desarrollo de *software,* es decir brindan soluciones probadas y documentadas acerca de problemas comunes en contexto. Esto implica simplificar esfuerzos en desarrollo y mantenimiento, contribuye a la consistencia del diseño y mejora la seguridad del *software*. (Harmes R. y Díaz D. 2008)

2.6.1 Singleton

En vísperas de la necesidad de que la aplicación posea una única instancia de una determinada capa que interactúa con sus adyacentes, se accede a este patrón para así poder controlar la capa en cuestión y asegurar una única instancia y que no se altere su código. Todo esto se logró mediante el objeto *module* propio del *framework* Express. (Harmes R. y Díaz D. 2008)

2.6.2 Proxy

En función de proporcionar un representante de una capa determinada del *software* y controlar el acceso a la misma el autor de esta investigación considero la utilización del patrón proxy, mediante el cual se mantiene una referencia y acceso a la capa en cuestión. Todo esto se logró mediante el objeto *exports propio* del *framework* Express. (Harmes R. y Díaz D. 2008)

2.6.3 Observador

En un entorno orientado a eventos, como son los navegadores web, donde se busca constantemente la atención de un usuario, el patrón de observador, también conocido como el patrón Publicador-Suscriptor. Este patrón se basa en que un objeto subscriptor está a la escucha de otro objeto publicador, que provee información de interés a los subscriptores, lo cual provee de independencia a los objetos. (Harmes R. y Díaz D. 2008)

Este patrón se aplica para manejar los eventos que le son necesarios a los usuarios subscritos en la aplicación, proporcionándole la información pertinente a los mismos.

2.7 Diagrama de Paquetes

Estructurar los elementos que intervienen en el *software* en paquetes ofrece una panorámica más amigable para el entendimiento del mismo, ya que permite observar el modelo en agrupaciones más simples como se puede ver en la figura 9.

Características

- Es usado como mecanismo para agrupar y encapsular elementos del UML permitiendo organizar dichos elementos y facilitando el manejo de los modelos del sistema.
- Posibilita fragmentar un modelo en aras de unir y empaquetar sus elementos en unidades individuales.
- Un paquete puede contener a uno o varios paquetes.
- Pueden existir dependencias ente los paquetes.
- Pueden tener una interfaz y se pueden emplear en la proyección del sistema a nivel macro.

(Mourillo R. 2015)

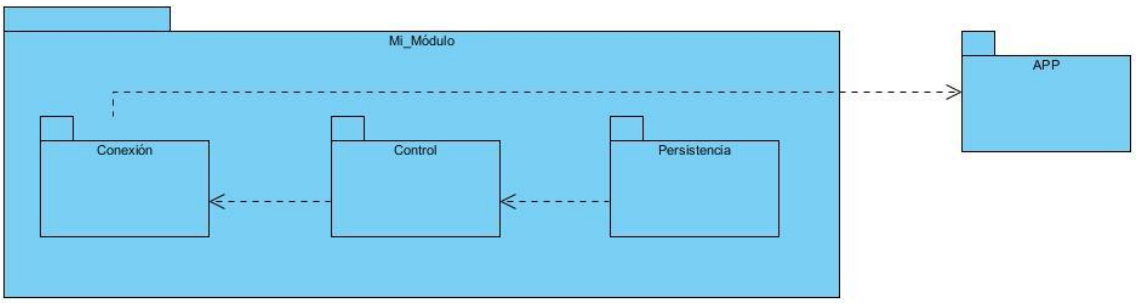


Figura 9. Diagrama de paquetes.

2.7.1 Descripción de los paquetes asociados a la solución propuesta

Paquete	Descripción
APP	Contiene los módulos referentes al manejo de la aplicación, así como los manejadores de error del servidor.
Mi_Módulo	Contiene todas las cabeceras(.js), correspondientes a las tres capaz de la aplicación.
Conexión	Contiene los módulos con los cuales se maneja la conexión mediante la tecnología WebSocket.
Control	Contiene la lógica de manejo de información entre los clientes conectados a la aplicación.
Persistencia	Contiene los módulos y lógica de manejo de acceso a la información que no pudo ser enviada a un cliente determinado por estar *offline*.

Tabla 8. Descripción del diagrama de paquetes.

2.8 Conclusiones del capítulo

En este capítulo se definió el modelo de dominio, el cual sienta la base para la solución propuesta. Además, se especificaron los requisitos del sistema que posibilitó la definición de los requisitos funcionales y no funcionales del módulo. También se logró una mejor comprensión de las funcionalidades del módulo a través de las historias de usuarios del sistema. Se tomó en cuenta una arquitectura basada en capas y como patrón de diseño se definió el Singleton, Proxy y Observador.

Capitulo III: Pruebas y validaciones del sistema

3.1 Introducción

En el presente capítulo se abordarán las pruebas necesarias para verificar el correcto funcionamiento del módulo de acuerdo a las pruebas unitarias y de integración, teniendo en cuenta el método de caja blanca. Además, se esbozaron los estilos de codificación, así como el modelo de despliegue en función de evidenciar las relaciones físicas entre los distintos subsistemas que componen la aplicación en tiempo de ejecución.

3.2 Modelo de despliegue

En función de evidenciar las relaciones físicas entre los distintos subsistemas (nodo) que componen la aplicación en tiempo de ejecución se realiza el modelo de despliegue (ver figura 10), donde un subsistema es un recurso de ejecución que varía entre un dispositivo de almacenamiento, procesamiento o acceso a la información. Los subsistemas se comunican mediante tecnologías bidireccionales lo que permite determinar el impacto de los gastos de asignación de recursos, así como las dependencias entre los mismos.

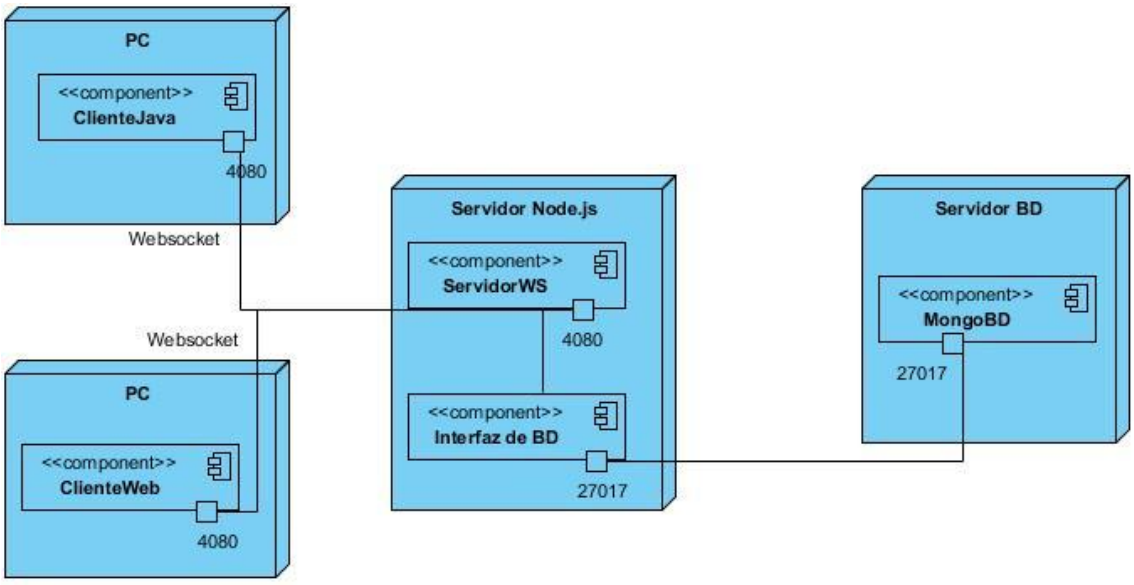


Figura 10. Diagrama de despliegue.

En el diagrama expuesto se observa como el hardware y el software trabajan de manera conjunta los clientes Web y Java dependen del servidor Node.js para comunicarse mediante la tecnología WebSocket por el puerto 4080 y a su vez este depende del servidor BD para garantizar la persistencia de los datos.

3.3 Estándares de codificación

Una de las herramientas indispensables que promueven la calidad del *software* es adquisición de modelos y estándares de codificación ya que dentro de sus diversas ventajas destaca la de lograr un estilo de código homogéneo asegurando su legibilidad, así como dotar de un marco para el encargado del mantenimiento/actualización del sistema, con código legible y bien documentado.

Indentación o sangrado: El código está indentado, de una forma adecuada y consistente (ver figura 11). La indentación (o sangrado) contribuye a ver la estructura del programa y propicia la identificación de posibles errores.

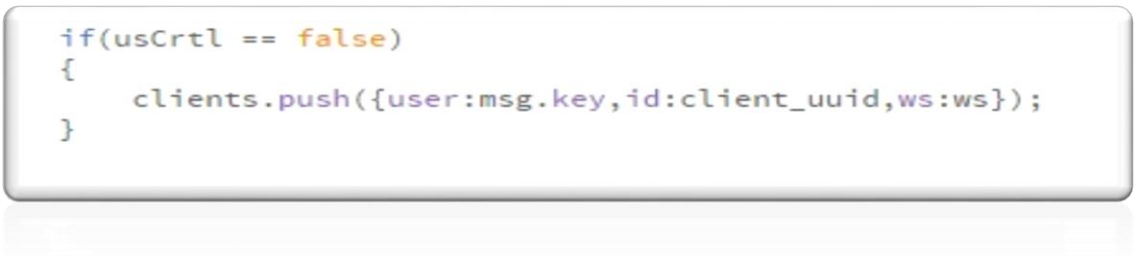


Figura 11. Ejemplo estilo de código en cuanto a identado.

Bloques: Se utiliza llaves para delimitar todos los bloques de sentencias de control y bucles (ver figura 12). Las llaves están en la misma columna (solos, sin código ejecutable).

```
for(i in clients)
{
    if(clients[i].id == client_uuid)
    {
        clients.splice(i,1);
        console.log('user disconnect',clients);

    }
}
```

Figura 12. Ejemplo estilo de código en cuanto a bloques.

Paréntesis: Se hace uso de los paréntesis para hacer más claro el orden de evaluación de operadores en una expresión (ver figura 13).

```
server.listen(port, function () { console.log('Listening on ' + server.address().port) });
```

Figura 13. Ejemplo estilo de código en cuanto a paréntesis.

Espacios en blanco: Se gestiona las líneas y espacios en función de hacer más claro y comprensible el código como se muestra en la figura 14. Así se conseguirá tener un código de fácil lectura (si fuera excesivamente compacto sería poco legible), y en el que sea sencillo localizar funciones y bloques de código.

```
ws.on('error', function(e)
 {

    console.log("error time",e);

 });
```

Figura 14. Ejemplo estilo de código en cuanto a espacios en blanco.

Comentarios: Los comentarios (ver figura 15) son minimizados (si se utilizan comentarios muy largos puede entorpecer la lectura), propicia la comprensión del código e incorporan información útil sobre las sentencias, bloques, variables, funciones que afectan.

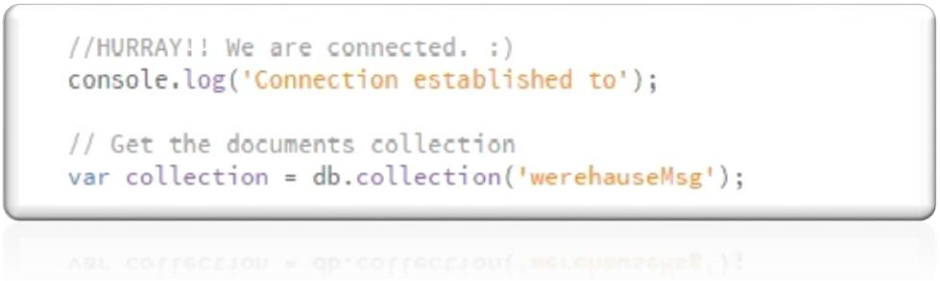


Figura 15. Ejemplo estilo de código en cuanto a espacios en comentarios.

3.4 Pruebas

Las pruebas de *software* son las investigaciones empíricas y técnicas en aras de descubrir errores no detectados que comprometan la calidad de dicho *software.* Este instrumento permite determinar el estado de calidad del producto gracias a un conjunto de actividades vinculadas al proceso de desarrollo que, dependiendo del modelo de prueba a realizar, se implementarán indistintamente del grado de desarrollo que posea el producto.

3.4.1 Niveles de Pruebas

Los niveles de prueba son diferentes puntos de vista con los que se jusga la calidad del *software* y que se abordan al lo largo de la vida del producto.Existen diversos niveles de pruebas donde se consideró las pruebas unitarias ya que posibilita al desarrollador medir las condiciones de robustez del código en cuestión, además de las pruebas de integración para identificar errores introducidos por la combinación de los módulos probados unitariamente.

3.4.2 Prueba unitaria

Método de caja blanca

El autor de la investigación consideró aplicar el método de caja blanca, en función de examinar el código fuente del *software* y en consecuencia a los diversos algoritmos y estrcturas de datos empleadas utilizando la técnica de camino básico. La aplicación de esta técnica certifica que todos los caminos dependientes en la ejecución han sido

puestos en cuestión por lo menos una vez así como adquirir una medidad de la complejidad del diseño procedimental. (Pressman R. 2001)

Camino básico

Es una prueba estructural que se deriva a patir del entendimiento del desarrollo del *software* y que posibilita determinar la comlejidad ciclomática de un fragmento de código. La complejidad ciclomática es una métrica que evidencia la complejidad lógica de un programa y proporciona una valor límite para el número de pruebas a diseñar. Esta complejidad se calcula de las tres formas siguientes:

- La complejidad ciclomática coincide con el número de regiones del grafo de flujo.
- La complejidad ciclomática, *V(G)*, de un grafo de flujo *G*, se define como
  V(G) = Aristas - Nodos + 2
- La complejidad ciclomática, *V(G)*, de un grafo de flujo *G*, también se define como
  V(G) = Nodos de predicado + 1

A continuación se muestra como se aplica dicha técnica a la Historia de Usuario “Almacenar los mensajes que no se pueden enviar debido a que el destinatario de una solicitud o de una respuesta no esté conectado” . El código fue particionado en bloques de ejecución, enumerados y que a su vez constituyen los nodos del camino básico como se muestra en la figura 16.

```
MongoClient.connect('mongodb://localhost:27017/prueba', function (err, db) {
          if (err) {//NODO 1
            console.log('Unable to connect to the mongoDB server. Error:', err);//NODO 2
          } else {//NODO 3
            console.log('Connection established to');//NODO 4
              var collection = db.collection('werehauseMsg');//NODO 4
              collection.insert(msgSend, function (err, result) {// NODO 4
                    if (err) {//NODO 5
                      console.log(err);//NODO 6
                    } else {//NODO 7
                      console.log('Inserted %d documents into the "werehauseMsg" collection. The documents inserted with
                                  "_id" are:', result.length, result);//NODO 8
                    }
                  });

          }
         db.close();//NODO 9
        });// NODO 10
```

Figura 16. Fragmento de código para aplicar la técnica de camino básico.

A continuación se muestra el grafo de flujo (ver figura 17) referente al código antes expuesto en función de calcular la complejidad ciclomática :

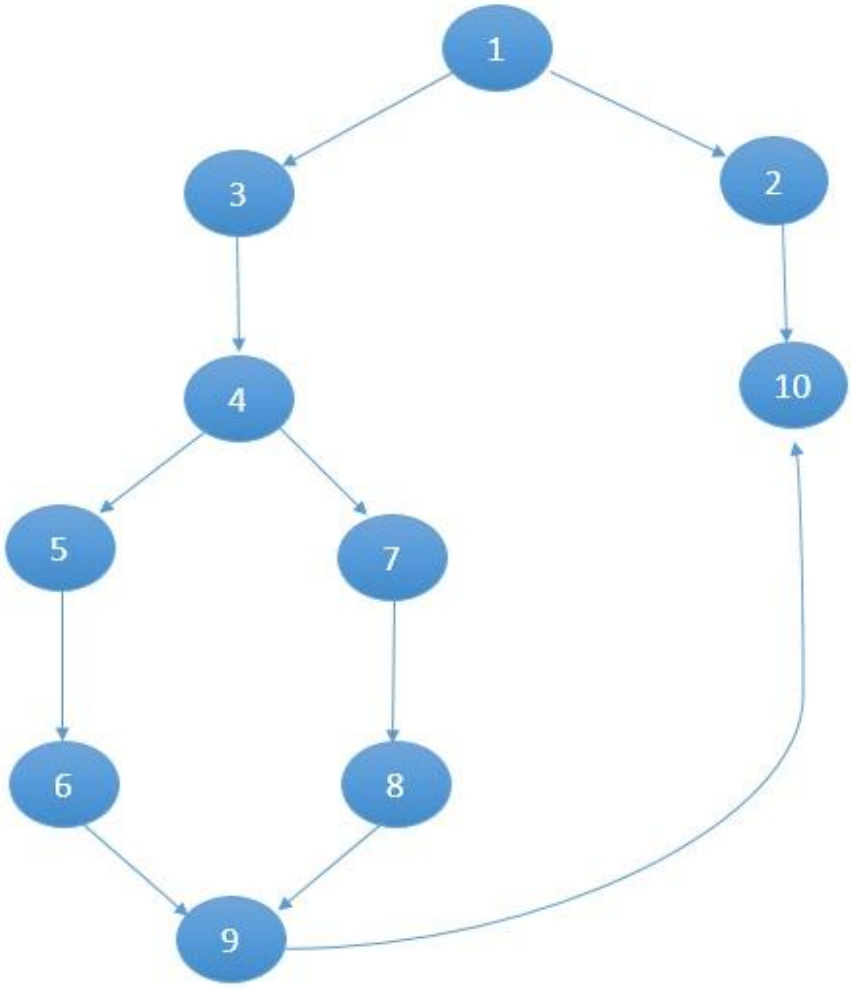


Figura 17. Grafo de flujo.

- Camino básico #1: 1 – 2 – 10.
- Camino básico #2: 1 – 3 – 4 – 5– 6 – 9 – 10.
- Camino básico #2: 1 – 3 – 4 – 7 – 8 – 9 – 10.

En la figura se muestra el resultado que se obtuvo al emprender la técnica de camino básico. Las aristas indican los posibles caminos a seguir a partir del nodo correspondiente. Con el camino básico precisado, se aplica una de las tres formas para calcular la complejidad ciclomática, se utilizó la fórmula V(G) = Aristas - Nodos + 2, la cual arrojó 11 artistas y 10 nodos, por lo tanto: V(G) = 11 - 10 + 2, evidenció V(G) = 3, además se empleó la fórmula V(G) = Nodos de predicado + 1, que evidenció V(G) = 2 + 1, quedando V(G) = 3, así como con la cantidad de regiones que son 3.

La complejidad ciclomática obtenida revela que existen 3 posibles casos de ejecución del algoritmo, por lo que se procede a ejecutar los casos de pruebas para cada uno de los caminos básicos determinados en el grafo de flujo,ver tablas 9,10 y 11:

Caso de Prueba Camino Básico #1
Descripción: La Variable “err” contiene los datos definidos para el mensaje de error en caso de que no se pueda acceder a la base de datos.
Condiciones: La base de datos MongoDB debe estar instalada.
Entrada: En la entrada de la dirección de la base de datos debe ser: • 'mongodb://localhost:27000/prueba'
Resultado Esperado: El sistema debe mostrar un mensaje de error de conexión con la base de datos.
Resultado: Satisfactorio.

Tabla 9. Caso de Prueba Camino Básico #1.

Caso de Prueba Camino Básico #2
Descripción: La Variable “err” contiene los datos definidos para el mensaje de error en caso de que no se pueda acceder a la base de datos.
Condiciones: La base de datos MongoDB debe estar instalada y bien configurada.
Entrada: En la entrada de la dirección de la base de datos debe ser: • 'mongodb://localhost:27017/prueba'
Resultado Esperado: El sistema debe insertar el mensaje que no se pudo enviar a un cliente determinado y cerrar la base de datos.
Resultado: Satisfactorio.

Tabla 10. Caso de Prueba Camino Básico #2.

Caso de Prueba Camino Básico #3
Descripción: : La Variable “err” contiene los datos definidos para el mensaje de error en caso de que no se pueda acceder a la base de datos.
Condiciones: La base de datos MongoDB debe estar instalada y bien configurada.
Entrada: En la función de insertar en la base de datos debe ser: • msgSend=[1,2.3;prueba-ya]
Resultado Esperado: El sistema debe mostrar un mensaje de error de inserción en la base de datos.
Resultado: Satisfactorio.

Tabla 11. Caso de Prueba Camino Básico #3.

3.4.3 Prueba de integración

Mediante este tipo de prueba se examinó la interacción entre las fachadas desarrolladas para el módulo de visualización, el módulo de recuperación de información mediante el módulo de interoperabilidad, esto evidenció una correcta comunicación entre estos componentes por lo que se asume estas pruebas como satisfactorias.

3.5 Resultados de las pruebas

En función de haberse realizado las pruebas funcionales mediante el método de caja blanca, se evidenció el correcto funcionamiento de la codificación del sistema.Todos los problemas detectados a raíz de las pruebas, fueron resueltos por el trabajo continuo del desarrollador, con un total de 5 no conformidades en la primera iteración,3 en la segunda y 1 en la tercera. A continuación se modelan los resultados en la tabla 12:

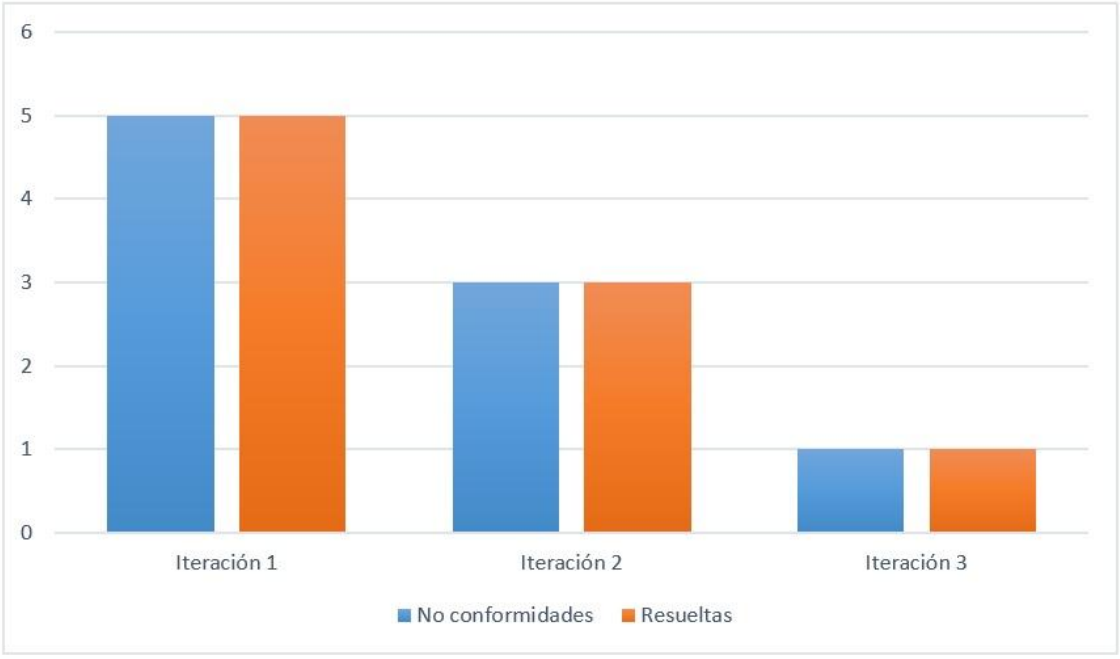


Tabla 12. Resultado de pruebas.

3.6 Conclusiones del capítulo

En este capítulo el autor de la investigación llevó a cabo un conjunto de pruebas al *software* de caja blanca con la técnica de camino básico donde cada dificultad encontrada en el desarrollo del producto fueron erradicadas en un total de 3 iteraciones así como las pruebas de integración. Además se mostró el modelo de despliegue y los estándares de codificación.

Conclusiones generales

- Se desarrolló un módulo que maneja la interoperabilidad entre los subsistemas del Cuadro de Mando Integral del SCADA Sainux posibilitando además la incorporación de demás módulos en un futuro.

Recomendación

Se recomienda cifrar los mensajes que se envían por la red ya que estos pueden ser capturados por algún *Sniffer* (analizador de paquetes) y posteriormente sea comprometida la información.

Bibliografía.

1. **AFUL.** Definición de la Interoperabilidad. [En línea] AFUL. [Citado el: 13 de Diciembre de 2015.] http://interoperability-definition.info/es/.

2. **Colouris, George.** 1994.

3. **Torre, Arturo Muñoz de la.** *INTRODUCCIÓN A NODE.JS A TRAVÉS DE KOANS.* Madrid : s.n., 2013.

4. **Pressman, Roger S.** *Ingenieria del software.Un enfoque práctico.* s.l. : Mc Graw Hill, 2001.

5. **César de la Torre Llorente, Unai Zorrilla Castro,Javier Calvarro Nelson,Miguel Ángel Ramos Barroso.** *Guía de Arquitectura N-Capas Orientada al Dominio con .Net 4.0.* s.l. : Krasis Consulting, S.L. www.krasis.com, 2010. ISBN: 978-84-936696-3-8.

6. **NEXTEL ENGINEERING SYSTEMS S.L.** NEXTEL ENGINEERING. [En línea] 2014. [Citado el: 1 de Noviembre de 2015.] http://www.nexteleng.es/.

7. **Galarza, Gabriela Paola Espinoza y Daniel Eduardo.** *Estudio de la aplicación de Inteligencia de Negocios en los procesos académicos caso de estudio"Universidad Politécnica Salesiana".* Guayaquil : Universidad Politécnica Salesiana, 2012.

8. **Sinnexus .** Cuadro de Mando Integral. [En línea] 2007-2016. [Citado el: 10 de Noviembre de 2015.] http://www.sinnexus.com/business_intelligence/cuadro_mando_integral.aspx.

9. **Páez, Francisco.** Cuadro de Mando Integral. [En línea] [Citado el: 5 de Diciembre de 2015.] http://www.cmigestion.es/servicios/consultoria-estrategica/cuadro-de-mando-integral/.

10. **Fernández, Alfonso.** *Indicadores de Gestión y Cuadro de Mando Integral.* s.l. : Instituto de Desarrollo Económico del Principado de Asturias.

11. **Aguilar, Carlos Proal.** Interoperabilidad. [En línea] [Citado el: 16 de Diciembre de 2015.] http://ict.udlap.mx/people/carlos/is346/admon10.html.

12. **Albentia Systems.** [En línea] [Citado el: 17 de Diciembre de 2015.] http://www.albentia.com/Docs/WP/ALB-W-000009spA2_VentajasEquipamientoAlbentia.pdf.

13. **Criteria Studio.** CommonJS para Titanium. [En línea] 11 de 7 de 2012. [Citado el: 2 de Octubre de 2015.] http://www.criteriastudio.com/commonjs-para-titanium/.

14. **Cuenca, Antonio Manuel Rubio.** [En línea] [Citado el: 15 de Octubre de 2015.] http://www.adminso.es/recursos/Proyectos/PFM/2010_11/PFM_CEL.pdf.

15. **Novas, Pablo.** fernetjs. [En línea] 22 de 11 de 2012. [Citado el: 10 de Octubre de 2015.] https://fernetjs.com/2012/11/websockets-y-socketio/.

16. **Chavez, Richard.** Sistemas Distribuidos. [En línea] 30 de 11 de 2015. [Citado el: 10 de 12 de 2015.] http://u201400389.blogspot.com/2015/11/websocket.html.

17. **Alarcón, José Manuel.** Fundamentos de bases de datos NoSQL: MongoDB. [En línea] 3 de Septiembre de 2014. [Citado el: 6 de Mayo de 2016.] http://www.campusmvp.es/recursos/post/Fundamentos-de-bases-de-datos-NoSQL-MongoDB.aspx.

18. **Guaita, Alvaro Martínez.** Brackets, el editor de código para la web. [En línea] 3 de 7 de 2012. [Citado el: 25 de Septiembre de 2015.] http://www.desarrolloweb.com/de_interes/brackets-editor-codigo-web-7163.html.

19. **González, Gabriela.** Los 10 mejores editores de texto para desarrolladores. [En línea] 11 de Octubre de 2013. [Citado el: 26 de Septiembre de 2015.] http://hipertextual.com/archivo/2013/10/mejores-editores-de-texto-para-desarrolladores/.

20. **Gutierrez, Dayron Limonta.** *Extensión del IDE Netbeans para el desarrollo de aplicaciones empleando el marco de trabajo Sauxe.* La Habana : s.n., 2012.

21. **Montesdeoca, Raquel Murillo.** Ingeniería de software. [En línea] 10 de Julio de 2015. [Citado el: 26 de Mayo de 2016.] http://jraquelm2.wix.com/ingenieriadesoftware#!-TEMA-9-DIAGRAMAS-DE-PAQUETES-Y-DE-SECUENCIAS/cmbz/55a540f80cf25b8bf7e9279f.

22. **Orallo, Enrique Hernández.** *El Lenguaje Unificado de Modelado (UML).*

23. **Rodriguez, Jorge.** *Pruebas unitarias.* 2006.

24. **Drake, José M.** Sistemas distribuidos de tiempo real. [En línea] [Citado el: 1 de Diciembre de 2015.] http://www.ctr.unican.es/asignaturas/procodis_3_II/Doc/Procodis_8_01.pdf.

25. **María, Jorge Abin De.** Taller de Interoperabilidad. [En línea] Septiembre de 2014. [Citado el: 16 de Octubre de 2015.] http://www.administracionpublica.gob.ec/wp-content/uploads/downloads/2014/10/Interoperabilidad-Introducci%C3%B3n-Mag.-Jorge-Albin.pdf.

26. **Dávila, Fernando.** LA INTELIGENCIA DEL NEGOCIO. [En línea] [Citado el: 20 de Octubre de 2015.] http://sigma.poligran.edu.co/politecnico/apoyo/cuadernos/intelligence.pdf.

27. **ORACLE.** ORACLE. [En línea] [Citado el: 26 de Septiembre de 2015.] http://www.oracle.com/ocom/groups/public/@otn/documents/webcontent/317529_esa.pdf.

28. **May, Fernando Pech.** Sistemas Distribuidos. [En línea] 2011. [Citado el: 19 de Octubre de 2015.] http://www.tamps.cinvestav.mx/~fpech/sd/files/modelos.pdf.

29. **Taipe, Ronald.** Sistemas distribuidos – Ventajas y Desventajas. [En línea] 16 de OCtubre de 2012. [Citado el: 30 de Octubre de 2015.] ronaldtaipe.blogspot.com/2012/10/sistemas-distribuidos-ventajas-y.html.

30. **JSON.** Introducción a JSON. [En línea] [Citado el: 3 de Diciembre de 2015.] http://www.json.org/json-es.html.

31. **GENBETA.** Programación funcional, un enfoque diferente a los problemas de siempre. [En línea] [Citado el: 10 de Diciembre de 2015.] http://www.genbetadev.com/paradigmas-de-programacion/programacion-funcional-un-enfoque-diferente-a-los-problemas-de-siempre.

32. **ETNASSOFT.** Entendiendo el tipado blando en Javascript. [En línea] 27 de Enero de 2016. [Citado el: 10 de Febrero de 2016.] http://www.etnassoft.com/2011/01/27/tipado-blando-en-javascript/.

33. **abcdisegno.** Funciones callback (Javascript y Jquery). [En línea] 2011. [Citado el: 10 de Noviembre de 2015.] http://www.abcdisegno.com/funciones-callback/.

34. **Muñoz, Rocio.** Node.js, javascript en servidor. [En línea] 24 de Octubre de 2014. [Citado el: 10 de Diciembre de 2015.] http://www.cantabriatic.com/node-js/.

35. **Dagoberto Montero, David B. Barrantes,Jorge M. Quirós.** Introducción a los sistemas de control supervisor y deadquisición de datos (SCADA). [En línea] 2004. [Citado el: 17 de Septiembre de 2015.] https://es.scribd.com/doc/13473499/Introduccion-a-Los-Sistemas-SCADA.

36. **OPC Unified Architecture.** Posibles usos y ventajas para fabricantes y usuarios de productos de automatización,TIC o MES. [En línea] [Citado el: 10 de Diciembre de 2015.] www.opcfoundation.org.

37. **Lombardi, Andrew.** *WebSocket.* s.l. : O'Reilly Media, Inc., 1005 Gravenstein Highway North, Sebastopol, CA 95472., 2015. ISBN: 978-1-449-36927-9.

38. **Grigorik, Ilya.** *High Performance Browser Networking.* s.l. : O'Reilly Media, Inc., 1005 Gravenstein Highway North, Sebastopol, CA 95472., 2013. ISBN: 978-1-449-34476-4.

39. **Seguin, Karl.** *The Little MongoDB Book.* 2014.

40. **Walls, Craig.** *Spring in Action 4th edition.* s.l. : Manning Publications Co. 20 Baldwin Road, Shelter Island, NY 11964, 2015. ISBN 9781617291203.

41. **Deitel.** *Java How to Program 7th Edition.* s.l. : © 2007 by Pearson Education, Inc.Upper Saddle River, New Jersey 07458, 2009. ISBN 0-13-222220-5.

42. **Larman, Craig.** *UML y Patrones. 2da Edicion.* 2003.

43. **Tejedor, Ramón Jesús Millán.** Programación de objetos distribuidos con CORBA. [En línea] 2006-2016. [Citado el: 12 de Noviembre de 2015.] http://www.ramonmillan.com/tutoriales/corba.php.

44. **Spruiel, Mark.** Client and Server Structure. [En línea] 21 de Agosto de 2015. [Citado el: 16 de Noviembre de 2015.] https://doc.zeroc.com/display/Ice36/Client+and+Server+Structure.

45. **ZeroC.** Ice, A Comprehensive RPC Framework. [En línea] 2015. [Citado el: 18 de Noviembre de 2015.] https://zeroc.com/.